Amiga™ Programs

Amazing Computing™

Volume 2 Number 5
U.S.A. $3.50
Canada $4.50

Commodore Amiga™ Information & Programs

Amiga™
DIGITIZED SOUND

AmigaBASIC™ Waveform Workshop

AMAZING REVIEWS
FutureSound™
PerfectSound™
SoundScape™ Sound Sampler

Programming A SoundScape Module in C
by Todor Fay

Amiga Hard Drive

Unbeatable SCSI Flexibility: No other Amiga hard drive can offer you: Capacities from 20MB to 760MB...plug-in compatibility with optical (WORM) drives, removable cartridge drives & CD ROMs...optional networking capability...dual drive compatibility with ST506 and add-ons, allowing use of up to 14 hard drives with your Amiga!

A REAL Track Record: C Ltd has been shipping Amiga hard drives since November, 1986. With thousands of units in use, you can count on C Ltd's proven hardware & software reliability.

Extraordinary Support: Call our technical support line, and you talk to the people who actually build the C Ltd products. Each drive is supplied with a complete technical manual. Each drive is *fully formatted*, with 10 MB of useful public domain software and commercial demo programs.

Cheaper By The Megabyte...

The **more** megabytes you buy, the **less** each magabyte costs you!

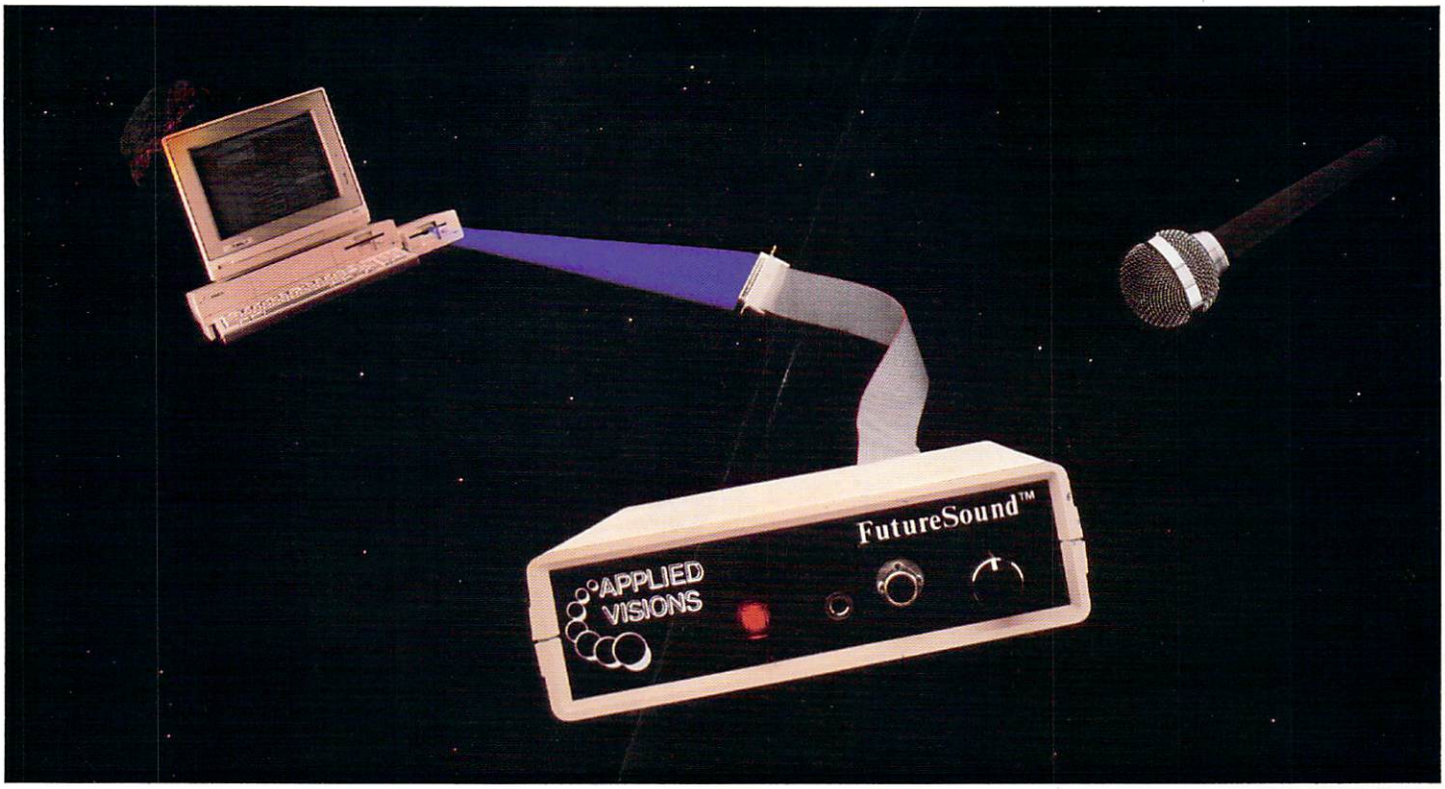
22MB	$ 999.95
33MB	1249.95
44MB	1499.95
50MB	1599.95
60MB	1999.95
80MB	2499.95
150MB	3299.95

additional sizes up to 750MB available upon request.

The Industry's Longest Warranty

One-year parts & labor warranty.

723 East Skinner / Wichita, Kansas 67211 / (316) 267-3807



"Open the pod bay doors, HAL..."

Programmers cast their vote!

Right now, leading software developers are hard at work on the next generation of Amiga® products. To add the spectacular sound effects we've all come to expect from Amiga software, they are overwhelmingly choosing one sound recording package... FutureSound. As one developer put it, "FutureSound should be standard equipment for the Amiga."

FutureSound the clear winner...

Why has FutureSound become the clear choice for digital sound sampling on the Amiga? The reason is obvious: a hardware design that has left nothing out. FutureSound includes two input sources, each with its own amplifier, one for a microphone and one for direct recording; input volume control; high speed 8-bit parallel interface, complete with an additional printer port; extra filters that take care of everything from background hiss to interference from the monitor; and of course, a microphone so that you can begin recording immediately.

What about software?

FutureSound transforms your Amiga into a powerful, multi-track recording studio. Of course, this innovative software package provides you with all the basic recording features you expect. But with FutureSound, this is just the beginning. A forty-page manual will guide you through such features as variable sampling rates, visual editing, mixing, special effects generation, and more. A major software publisher is soon to release a simulation with an engine roar that will rattle your teeth. This incredible reverberation effect was designed with FutureSound's software.

APPLIED VISIONS

Question: What can a 300 pound space creature do with these sounds?

Answer: Anything he wants.

Since FutureSound is IFF compatible (actually three separate formats are supported) your sounds can be used by most Amiga sound applications. With FutureSound and Deluxe Video Construction Set from Electronic Arts, your video creations can use the voice of Mr. Spock, your mother-in-law, or a disturbed super computer.

Programming support is also provided. Whether you're a "C" programming wiz or a Sunday afternoon BASIC hacker, all the routines you need are on the non-copy protected diskette.

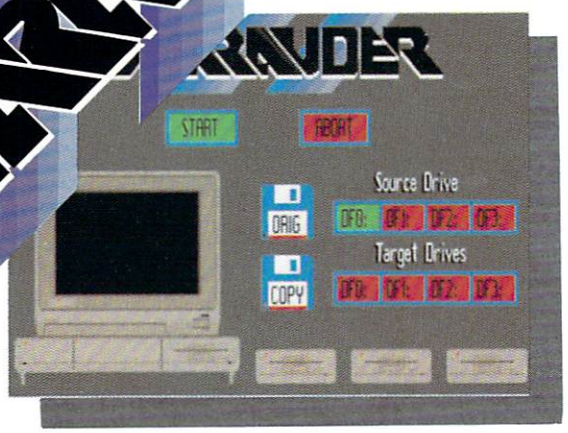
Your Amiga dealer should have FutureSound in stock. If not, just give us a call and for $175 (VISA, MasterCard or COD) we'll send one right out to you. Ahead warp factor one!

Applied Visions, Inc., Suite 2200, One Kendall Square
Cambridge, MA 02139 (617) 494-5417

The Mirror Copier Can Now Back Up A Disk Almost As Fast As Marauder II,

And It Only Costs About 25% More!

MARAUDER®



SEE FOR YOURSELF

	Marauder II	Mirror
Duplication Speed	83 Sec.	1 Minute 48 Sec.
Upgradable With Strategy Files	YES	NO
Mouse Driven User Interface	YES	NO
Exit Without Restarting Amiga	YES	NO
Runs From Workbench or CLI	YES	NO
Makes Multiple Simultaneous Copies From One Original	YES	NO
Copies Itself	YES	NO
Copies The Mirror	YES	NO
Price	**$39.95**	**$49.95**

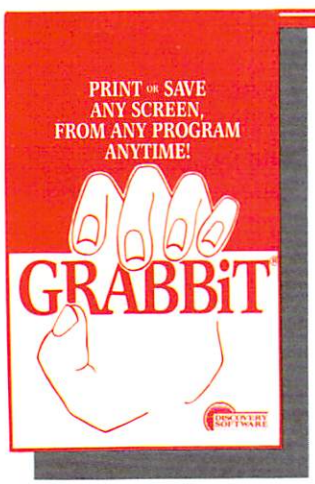
Marauder II is the most powerful copier ever produced for Amiga. It will automatically copy ALL software released to date , and it requires no hardware modification of any kind. It produces completely unprotected copies of most Amiga software faster and better than any other copier.

No other copier can copy as much software as Marauder II.

Marauder II also has the most advanced user interface your money can buy. If you have an Amiga you already know how to use Marauder II. You never have to reboot your machine to use Marauder II, it is completely compatible with the Amiga's multitasking operating system.

Marauder II has been designed with your future needs in mind. As protection schemes change you can update the program yourself with our unique "Strategy Files." The Strategy Files are developed as new software is released so that you can get them quickly and easily when you need them.

Compare the features of Marauder II to our competition and you'll see that Marauder II is quite simply the best copier you can get, at any price! And for only **$39.95** you can rest assured that your software investment is safely protected against damage, loss or theft. Don't wait, order now!



NOW YOU CAN SAVE ANY SCREEN, FROM ANY PROGRAM, ANYTIME WITH GRABBIT.

With GRABBIT you can capture exactly what you see on your screen in an instant, regardless of what programs you're running. GRABBIT works with all video modes, including "Hold and Modify." What's more, GRABBIT runs completely in the background, transparent to your other software. GRABBIT is always ready for you to use, even when you're in the middle of another program. As if that is not enough, GRABBIT requires only about 10K RAM to operate, and it supports dozens of printers. GRABBIT is truly a productivity power tool for your AMIGA!

GRABBIT is far superior to other screen-printing "programs" because of its small size and quick performance. No complicated setup is required, just install and go! Also, GRABBIT doesn't require the screen to remain visible during printing or saving, and stopping the print operation is as easy as starting it.

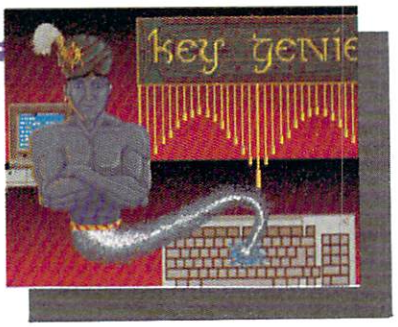
GRABBIT supports all standard Amiga printer drivers. GRABBIT also supports full color printing.

In addition to GRABBIT's printing capabilities, the package also includes a powerful utility program "ANYTIME." The ANYTIME bonus program is a "Preferences" style palette requester that allows you to change any colors of any screen, anytime! With ANYTIME, you are NOW capable of customizing all colors to match your printer's hardcopy to the screen's colors.

Once you start using GRABBIT and the bonus program ANYTIME you will want it on every disk. You get all the power of this sizzling new software for an unbelievably low **$29.95** + $5 shipping and handling.

With Key Genie — One Key Launches 1000 Strokes!

This amazing keyboard macro processor is just what you need to give your fingers a rest. The Genie is always at work to save you time and keystrokes. Complicated or repitious keyboard sequences are easily assigned to a key you choose through the Genie's Pop-Up Macro Definition Window. You can also load and save your favorite macro sequences on disk. Once saved, the macros can be automatically installed at startup to save time. In addition to the Genie's powers, Discovery Software has added a bonus program "Turbo-Shell". The Shell is an AmigaDOS performance enhancer that you shouldn't be without! The Shell gives you the capability to recall previous CLI commands with the arrow keys so that mistyped commands can be quickly corrected, and frequently used commands can be easily repeated. Fast AmigaDOS command replacements give you UNIX-style performance from your Amiga.
What other software does so much for you at such a low price. Only **$49.95** + $5 shipping and handling.



When ordering from overseas add an additional $5.00 shipping fo first class airmai

11-1-86

DISCOVERY SOFTWARE INTERNATIONAL

903 E. Willow Grove Ave., Wyndmoor, PA 19118 (215) 242-4666

Amazing Dealers

The following are Amazing Dealers, dedicated to supporting the Commodore-Amiga™.
They carry Amazing Computing™, your resource for information on the Amiga™.

If you are not an Amazing Dealer, but would like to become one, contact:

PiM Publications, Inc.
P.O.Box 869
Fall River, MA. 02722
1-617-678-4200

Alabama
Alabama Book Store — Tuscaloosa
Abax Data Systems — Huntsville
Command Computers — Birmingham
Electronics Boutique — Hoover
Madison Books & Computers — Madison
Mail Tel Communications — Mobile
Mel's Photo & Computer Shop — Montgomery
Universal Computer System — Mobile

Arkansas
SIS Inc. — Mt. Home
The Micro Shop — Little Rock

Arizona
Computer Library, Inc. — Tempe
Computer West — Phoenix
Computer Works — Glendale
Copperstate Business Systems — Phoenix
Copperstate Computers — Tucson
Hacks's — Tempe
North American Digital — Tucson

California
Ami Link — San Rafael
Augnet — Freemont
Balboa Office Supply — San Diego
Bits n Pieces Comp Prod. — Anahiem
Brown Knows Computing — Redlands
Byte Factory Computers — Salinas
C.F.T. — Claremont
Century Computer Systems — La Habra
Circus of Books II — LosAngeles
Commotech — National City
Compare Inc. — Ridgecrest
Computer Attic — Palo Alto
Computer Literacy Bookshop — Sunnyvale
Computer Nook, Inc. — San Bernadino
Computer Outlet — San Diego
Computer Outlet — San Marcos
Computer Time — Citrus Hts
ComputerLand Freemont — Freemont
ComputerLand of Stockton — Stockton
Dasophile — Watsonville
De Lauer's News Agency — Oakland
Electronics Boutique — West Los Angeles
Empire Computers — Santa Rosa
Harding Way News — Stockton
Home Computing Center — Concord
Home Computing Center — Mill Valley
Home Computing Center — San Leandro
Home Computing Center — San Bruno
HT Electronics — Sunnyvale
K J Computers — Granada Hills
Levity — North Hollywood
Micro Flight — San Jacinto
Nonagon Software — El Cerrito
Op Amp Tech Books — Los Angeles
World Book & News — Hoollywood
R&R Software — Chula Vista
R&R Software — San Diego
Ridgecrest Computer Ctr. Inc. — Ridgecrest
S.O.S. Computers — Los Angeles
Software First — Santa Rosa
Software Central — Pasadena
Software Plus — Citrus Heights
Tower Books — Chico
Tower Books — Citrus Hights
Tower Books — Concord
Tower Books — Mountain View
Tower Books — Sacramento
Tower Books — San Mateo
The Computer Room — Scotts Valley
Winner's Circle — Berkley

Colorado
Citadel Computers — Colorado Springs
Computer Discount — Denver
Computer Room — Aurora
Ebert Personal Computers — Aurora
Micro Computer World — Fort Collins
Micro World — Lakewood
Sun Country Computers — Colorado Springs
Aurora Newsstand — Aurora
Whole Life Distributors — Englewood
Whole Life Distributors — Englewood

Connecticut
Amiga Business Computer — Norwalk
Computer Outlet — PlsVernon
Electronics Boutique — Danbury
Friendly Computers — Plantsville
Mnemonics Inc. — Norwalk
Personal Computer center — Norwich
Software City — Orange
Software Kingdom — East Windsor
Spectrum Computers — New Haven

Delaware
Castle Video — Newark

Florida
A A Computers — Jacksonville
Book Mania — Orlando
Commodore Computer Shop — Ocala
Computer Bar — Pennsacola
Computer Base — Brandon
Computer Image — Miami
Computer Terminal Inc. — Sebring
Computers 4 Rent — W. Palm Beach
Computers Plus — South Daytona
Cotronics Inc. — Stewart
Education Computers Etc. — Tallahasse
Electronics Boutique — Boynton Beach
Electronics Boutique — Tallahassee
Electronics Boutique — Miami
Electronics Boutique — Orlando
Family Computers — Sarasota
Florida Book Store — Gainsville
Games 'N' Gadgets — Orange Park
Games 'N' Gadgets — Jacksonville
Games 'N' Gadgets — Tallahassee
Games 'N' Gadgets — Clearwater
Gulf Coast Computer Exchange — Panama City
Heartland Computers — Mulberry
K&L Computers — E. Palatka
Megaport Computer Center — South Venice
Micro Computer Center — Largo
Micro's Etc. — Altamonte Springs
Now Age Electronics — St. Petersburg
Random Access Computers Inc. — Ft. Walton Beach
Software's Computer Center Inc. — Holly Hill
Sunny's at Sunset Inc. — Sunrise
The Open Door — Cocoa
The PC Collection — Coral Gables

Georgia
Electronics Boutique — Atlanta
Electronics Boutique — Atlanta
Electronics Boutique — Kennesaw
Future System — Augusta
Games 'N' Gadgets — Atlanta
Games 'N' Gadgets — Atlanta
Michael Donnellan — Decatur
Software South Inc. — Savannah
The 64 Store — Atlanta

Hawaii
Astro Computers & Electronics — Aiea
Computer House Inc. — Honolulu
J & C Repair — Aiea

Idaho
A.B.I. Computer & Video — Boise
ABI Computer & Video — Nampa

Illinois
A Bitokster — Rockford
Computerland O'Hare — Park Ridge
ComputerLand of Niles — Niles
DigitalWorld — Addison
I.C.V. Data Center Inc. — Mundelein
Illini Microcomputers Inc. — Naperville
Interstate Computer Service — LeMont
Krooh's & Brentano's — Chicago
Micro Age — Oakawn
Software Or Systems — Fairview Hts.
Software Plus — Wheeling
The Memory Expansion — Normal
Unique Computer — Sterling

Indiana
Burkat Computer Ctr. — South Bend
Bytrex Computer — Ft. Wayne
Bytrex of Angola — Angola
Computer Corner — Marion
Computer Corner — Fort Wayne
Computer People Inc. — Michigan City
Greg Chaney — Greenfield
Micro Computer Centers Inc. — Indianapolis
Spectra Computer Sys. Inc. — Evansville
Vons Computers — West Lafayette

Iowa
Century Systems — Des Moine
Grone: Computers — Des Moine
Micro Computer Applications LTD — Marshalltown

Kansas
J & L Electronics — Liberal
Micronexus Computers — Newton
Midwest Computers — Manhattan
Thorobred Computers — Topeka

Kentucky
Computer Dimensions Inc. — Lexington
MicroAge Computer Store — Lexington
Reality World Computers — Lexington
The Software Source — Louisville

Louisiana
Computer Time — Shreveport
Electronics Boutique — Kenner
Electronics Boutique — Monroe
Electronics Boutique — Lawrenceville
Modern Business Machines — Metairie
Software Center International — Metairie
Software Solutions — Baton Rouge
Software Source — Lafayette
The Computer Clinic — Lafayette

Maryland
Buried Treasure — Rockville
Computer Crafters — Wheaton
Compuvision Vision Center — Columbia
Electronics Boutique — Baltimore
Electronics Boutique — Baltimore
Electronics Boutique — Anne Arundel County
Electronics Boutique — Owings Mills
Games 'N' Gadgets — Baltimore
Games 'N' Gadgets — Baltimore
Games 'N' Gadgets — Columbia
Layhill Newsstand — Kensington
Micro Computer Center — Baltimore
Software Advantage — Rockville
Software N Things — College Park
Waldorf Computer — Waldorf

Massachusetts
Club Computer — Cambridge
E.U. Wurlitzer Co. — Boston
Electronics Boutique — Holyoke
General Computer Store — Framingham
HCS Computer Center — Marshfield
LCA Video & Computer Ctr. — Norwood
Memory Location — Wellsley
OmniTek Computers — Tewksbury
Pioneer Valley Data Equipment — Amherst
The Bit Bucket — West Newton
The Software Haus — Quincy
Tycom Inc. — Pittsfield

Maine
Computer Barn — Damariscotta

Michigan
Basic Business Control Sys. — Traverse City
Basic Computer Center — Grand Rapids
Bits-Bytes-Nibbles — Petoskey
Canton Computer — Canton
CompuSoft — Kalamazoo
Direct Access — Lansing
Edmunton Computer Center — Pontiac
Galaxy Computers — Allen Park
Global Computer Center — Dearborn
I O Software — Flint
Michigan Software — Novi
Midland Computer Shop — Midland
Central Park Computers — Okemos
Programs Unlimited — Grand Rapids
Roseville Computer Store — Warren
Slip Disk — Madison Heights
Software Plus — Grand Rapids
Specialists Computer — Livonia
State Street Computers — Ann Arbor
T.J. Enterprises — Clinton
The Software House — Kalamazoo
Ye Olde Computer Shoppe — Ypsilanti

Minnesota
Computers, etc. — Eagan
Computer Outfters — Roseville
Computer Place — Minneapolis
JMH Software — Maple Grove
MCD of Hibbing Inc. — Hibbing
Midwest Computer Services — Clearwater
OnLine Computers — Minnetonka
Specialists IN — Hopkins
Valiant Inc. — Stillwater

Mississippi
Electronics Boutique — Ridgeland
Enterprises Unlimited — Jackson
National Computer Center — Ocean Springs

Missouri
Assisted Computer Services — Springfield
Brands Mart Computers — Kansas City
Data Byte Computer Co. — St. Louis
Instant Replay Ltd. — Chesterfield
Systems Plus of St. Louis — St. Louis

Montana
Appelgren Computer Systems — Great Falls

Nebraska
BULLS-EYE — Lincoln
My Com Inc. — Plattsmouth
Double E Electronics — Omaha

Nevada
Century 23 Inc. — Las Vegas
Computer World — Las Vegas
ComputerLand of Carson — Carson City
Tower Records — Las Vegas

New Hampshire
Computer Mart of New Hampshire — Nashua
Diversified Computers — Keene
Electronics Boutique — Nashua

New Jersey
Electronics Boutique — Voohrees
Electronics Boutique — Wayne
Electronics Boutique — Rockaway
Electronics Boutique — Cherry Hil
Electronics Boutique — Woodbridge
Family Computer Centres — Fairfield
Family Computer Centres — South Orange
Family Computing Centres — Ridgewood
Games 'N' Gadgets — Burlington
Games 'N' Gadgets — Livingston
Morris County Stationers — Chester

New Mexico
Academy Computers — Albuquerque
CVM Systems — Santa Teresa
New Horizon Computer System — Alamogordo
Page One Newsstand — Albuquerque
Technological Concepts — Las Cruces

New York
Amuse — New York
Amicus- Hudson Valley — Wappingers Falls
Amiga Business Computer — North Port
Byte Shop — Merrick
CIA Software Center — Flushing
Compurite Center — White Plains
Computer Outlet — Jamestown
Computers Etc. — Syracuse
Cromland Inc. — Brooklyn
Electronics Boutique — Lake Grove
Electronics Boutique — Huntington Station
Electronics Boutique — Albany
Electronics Boutique — Valley Stream
Games 'N' Gadgets — Nanuet
Games 'N' Gadgets — Garden City
Leigh's Computers — New York
Mnemonics — North Port
Software Centre International — Forest Hills
Software & Such — Scotia
Software City — Forest Hills
Software Supermarket — Kenmore
Star Tech Systems — Massena
Syntaire Trading — New York
Tadiran — New York
Tedrow Business Products — Rochester
The Computer Cellar — Albany
The Working Computer — Stoney Brook
Video Computer Bin — Brooklyn
Video Computer Center — Rome
Video Connection — Schenectady
Village Computer Inc. — New York
World Computers Inc. — Hicksville

North Carolina
Digitz — Raleigh
Games 'N' Gadgets — Raleigh
Software City — Raleigh
Software City — Charlotte
TDS Computers — Durham
TDS Computers — Durham
Triad Computers — Greensboro

North Dakota
Computer Associates — Fargo
G.F. Computers — Grand Forks
The Computer Store — Minot

Ohio
Computer Network — Newark
Computer Site — Cleveland
Computers Plus of Ohio — Upper Sandusky
Earthrise Micro Systems — Columbus
Electronics Boutique — Columbus
Fairborn Home Computer — Fairborn
Games 'N' Gadgets — North Randall
Lakes Consumer Electronics — Akron
Main Office Supply — Coshocton
Micro Center — Columbus
Microwave Magic — Fairfield
North Coast Programming — Willoughby
Personal Computer Co. — Youngstown
Program Store — Columbus
Quality Computer Appliances — Toledo
Quality Computer Appliances — Toledo
Stanton Computer Center — Sandusky
Software Centre International — North Olmsted
Software and More — Cincinnati
Software City — Centerville
Software Connection — Mayfield Hts.

Oklahoma
Andromeda — LaGrande
Colonial Video & Computer — Bartlesville
Micro Link Computer Store — Tulsa
P C Tech — Tulsa
Second Hand Software — Oklahoma City
Video-Comp Inc. — Lawton

Oregon
Andromeda — LaGrande
BTE Telephone & Computer — Klamath Falls
Clackamas Computers — Beaverton
Clackamas Computers — Clackamas
Computron Business Systems — Portland
I3 Computers — Portland
Software Express — Eugene

Pennsylvania
Alpha Omega EBE — Warren
Basic Computer Systems — Hermitage
Circle Computer — Ephrata
Computers and Games — Reading
Data Seftique Computerware — Pittsburg
Downington Computer Center — Downington
East Coast Software — Hershey
Electronics Boutique — North Wales
Electronics Boutique — Langehorne
Electronics Boutique — Exton
Electronics Boutique — Norristown
Electronics Boutique — Pittsburgh
Electronics Boutique — King of Prussia
Electronics Boutique — Whitehal
Games 'N' Gadgets — Lancaster
Games 'N' Gadgets — King of Prussia
Games 'N' Gadgets — Media
Games 'N' Gadgets — West Mifflin
Genet's Books — King of Prussia
North Central Digital Systems — Danville
Pitsburgh Computer Store — Pittsburgh
The Electronic Boutique — Norristown
Triangle Computers — Indiana

Rhode Island
Micro Limits — Smithfield
International Computer Services — Johnston
Micro Limits — Warwick

South Carolina
Avcom International Inc. — West Columbia
Electronics Boutique — Greenville
Horizon — Greenville
Micro Computer Depot — Sumter
Software Solutions — Westwood Plaza
Watson's Jewelery — Chesterfield

South Dakota
Computer Business Center — Rapid City

Tennessee
ComputerLab of Memphis — Memphis
Games 'N' Gadgets — Memphis
Software First — Nashville

Texas
A B. Computer — Round Rock
Byte Shop — Austin
C.E.M. Corporation — Brazoria
Colonial Video — Houston
Computer Age — Houston
Computer Age — Dallas
Computer Magic — Austin
Computer Revelations — Houston
Computer Time Inc. — Marshal
DataPro — Wichita Falls
FNS Computer Services — Texarkana
Metropolitan Computer — Richardson
Micro Search — Houston
Regency Educational Systems — Dallas
Softex — Watauga
Software Annex — Houston
Software Library — Wichita Falls
Software Terminal — Fort Worth
The Computer Experience — San Antonio
The Computer Stop — Abilene

Utah
Computers Plus — Murray

Vermont
Ray Supply Inc. — South Burlington

Virginia
Colony Ltd. — Hampton
Diskovery — Falls Church
Family Computer Center — Fairfax
Games 'N' Gadgets — Hampton
Games 'N' Gadgets — Virginia Beach
Games 'N' Gadgets — Virginia Beach
Software Central — Norfolk
Software City — Richmond
Virginia Micro Systems — Woodbridge

Washington
A.P.P.L.E. CO-OP — Renton
Apple Valley Computers — E. Wenatchee
Bull Dog News — Seattle
Byte World — Kent
Com-Soft Computers — Everett
Computer & Movie Magic — Yakima
Computers + Inc. — Tacoma
Family Computers — Lynwood
Kitsap Amiga Operators — Bellview
Micro Age Computer Store — Bellview
Micro Specs Inc. — Spokane
Nibble Shs — Tacoma
Omni International — Seattle
PC Tech — Kirkland
Programs Plus — Seattle
Take-A-Byte — Kennewick

West Virginia
Layhill Newsstand — Charleston

Wisconsin
Celarian Computers — Racine
Computer Software Center — Milwaukee
Fox Valley Personal Computer — Oshkosh
Mam's Computers — Stevens Point
Software Shoppe — Fond Du Lac
TMW Software — Greenbay
TMW Software Inc. — Wausau

Wyoming
Byte Emporium — Laramie

Canada
Compute or Play — Alberta
Computerworks — Alberta
Computer Shop — Alberta
Data Pro — Alberta
Computer Shop of Calgary — Alberta
The Software House — Alberta
Hindsin Computing — Alberta
Red Deer Micro Systems — Alberta
Memory Land — Alberta
Computer King — British Columbia
Corel — British Columbia
Nutec — British Columbia
Sprita — British Columbia
Strider — British Columbia
Super Software — British Columbia
Canadian Computer Era — Manitoba
Micromart — Manitoba
ComputerWorld — Manitoba
J.W. Allen — Manitoba
Kobotek — Manitoba
Minicomp — Manitoba
Arkon — Ontario
Chatam Computer — Ontario
Computerland — Ontario
Compsec — Ontario
Electronics 2001 — Ontario
Hastings Data — Ontario
The Computer & You — Ontario
Thornhil — Ontario
Software Co. — Quebec
Deltronic Solare — Quebec
Microdata — Quebec
Info Mercier — Quebec
Laval Micro — Quebec
Vision Tronique — Quebec
Distribution Vielpro — Quebec

Distributed in Canada by
Avtra Software Distributors — Concord, Ontario
Compulit Distributors — British Columbia
Phase 4 Dist. Inc. — Calgary, Alberta

Argentina
Logic Computer — Buenos Aires

Australia
High Technology Computer Supplies — Brighton, Melborn
Maxwell Office Equipment — Abbotsford, Melburn

Japan
Pineapple 6502 — Tokyo, 111

Saudia Arabia
Abdulla Fouad & Sons — Daman

Switzerland
Software Land AG — Zuerich

Amazing Computing™ is also available in most B. Dalton Booksellers stores, B. Dalton Software stores, and Software Etc. locations.

Amazing Computing™

Volume 2 Number 5

Amazing Reviews

*...also see **AmigaNotes** and **More AmigaNotes** for more reviews.*

Amazing Columns

Amazing Contents

Volume 2 Number 5

Amazing Features

Amazing Columns

"Your Resource to the Commodore Amiga™"

Amazing Computing™

Your Resource to the Commodore Amiga™

Publisher: Joyce Hicks
Circulation Manager: Doris Gamble
Assistant to the Publisher:
Robert James Hicks
Traffic Manager: Robert Gamble

Managing Editor: Don Hicks
Assistant Editor: Ernest P. Viveiros Jr.
Hardware Editor: Ernest P. Viveiros
Amicus & Technical Editor: John Foust
Music Editor: Richard Rae
Art Director: Keith Conforti
Assistant Advertising Manager:
John David Fastino
Production Manager: Mark Thibault
Assistant PM: Keven Desmarais
Copy Editor: Michael Cabral

Advertising Sales
&
Editorial

1-617-678-4200

Amazing Computing™ (ISSN 0886-9480) is published by PiM Publications, Inc. P.O. Box 869, Fall River, MA. 02722. Subscriptions: in the U.S. 12 Issues for $24.00; Canada and Mexico, $30.00; Overseas, $35.00. Printed in the U.S.A.

First Class or Air Mail rates available upon request.

PiM Publications, Inc. maintains the right to refuse any advertising.

PiM Publications Inc. is not obligated to return unsolicited materials. All materials requesting return must be received with a Self Addressed Stamped Mailer.

MOVING?

Please advise PiM Publications Inc. at least four weeks before you move so your magazines will be delivered to your new address.

Software designed for AMIGA®.

Lattice® C Compiler — *$225.00*

New version 3.1 of the AMIGA DOS C Compiler replaces version 3.03. Major enhancements include the addition of: TMU, an assembler, a faster linker and version 3 MS-DOS.

With more than 30,000 users worldwide, Lattice C Compilers set the industry standard for MS-DOS software development. Lattice C gives you all you need for development of programs on the AMIGA. Lattice C is a full implementation of Kernighan and Ritchie with the ANSI C extensions and many additional features.

Professional Lattice® C Compiler — *$375.00*

A new product called the Professional Lattice C Compiler is now available. It includes the C Compiler package (complete with TMU), plus LMK, LSE and the Metascope Debugger.

AMIGA® C Cross Compiler — *$500.00*

Allows AMIGA development on your MS-DOS system. Price includes the Professional Lattice C Compiler described above.

Lattice Screen Editor (LSE™) — *$100.00*

Designed as a programmer's editor, *Lattice Screen Editor (LSE)* is fast, flexible and easy to learn. *LSE's* multi-window environment provides all the editor functions you need including block moves, pattern searches and "cut and paste." In addition, *LSE* offers special features for programmers such as an error tracking mode and three Assembly Language input modes. You can also create macros or customize keystrokes, menus, and prompts to your style and preferences.

Lattice dBC III™ Library — *$150.00*

The *dBC III library* lets you create, access and update files that are compatible with Ashton-Tate's dBASE system. *dBC III's* C functions let you extend existing dBASE applications or allow your users to process their data using *dBC III* or dBASE III.

Lattice Text Utilities (TMU™) — *$75.00*

Lattice Text Utilities consists of eight software tools to help you manage your text files. GREP searches files for the specified pattern. DIFF compares two files and lists their differences. EXTRACT creates a list of file names to be extracted from the current directory. BUILD creates batch files from a previously generated file name list. WC displays the number of characters and optionally the checksum of a specified file. ED is a line editor which can utilize output from other *TMU* software in an automated batch mode. SPLAT searches files for a specified character string and replaces every occurrence with a specified string. And FILES lists, copies, erases or removes files or entire directory structures which meet the specified conditions.

Lattice Unicalc® Spreadsheet — *$79.95*

Unicalc is a simple-to-operate program that turns your AMIGA computer into an electronic spreadsheet. Using *Unicalc* you can easily create sales reports, expense accounts, balance sheets, or any other reports you had to do manually.

Unicalc offers the versatility you've come to expect from business software, plus the speed and processing power of the AMIGA.

• 8192 row by 256 column processing area • Comprehensive context-sensitive help screens • Cells can contain numeric, algebraic formulas and titles • Foreign language customization for all prompts and messages • Complete library of algebraic and conditional functions • Dual window capabilities • Floating point and scientific notation available • Complete load, save and print capabilities • Unique customization capability for your every application • Full compatibility with other leading spreadsheets • Full menu and mouse support.

Lattice MacLibrary™ — *$100.00*

The *Lattice MacLibrary™* is a collection of more than sixty C functions which allow you to quickly and efficiently take advantage of the powerful capabilities of the AMIGA.

Even if your knowledge of the AMIGA is limited, *MacLibrary* can ease your job of implementing screens, windows and gadgets by utilizing the functions, examples and sample programs included with the package.

Other *MacLibrary* routines are functionally compatible with the most widely used Apple® Macintosh™ Quickdraw Routines™, Standard File Package and Toolbox Utility Routines enabling you to rapidly convert your Macintosh programs to run on the AMIGA.

Panel™ — *$195.00*

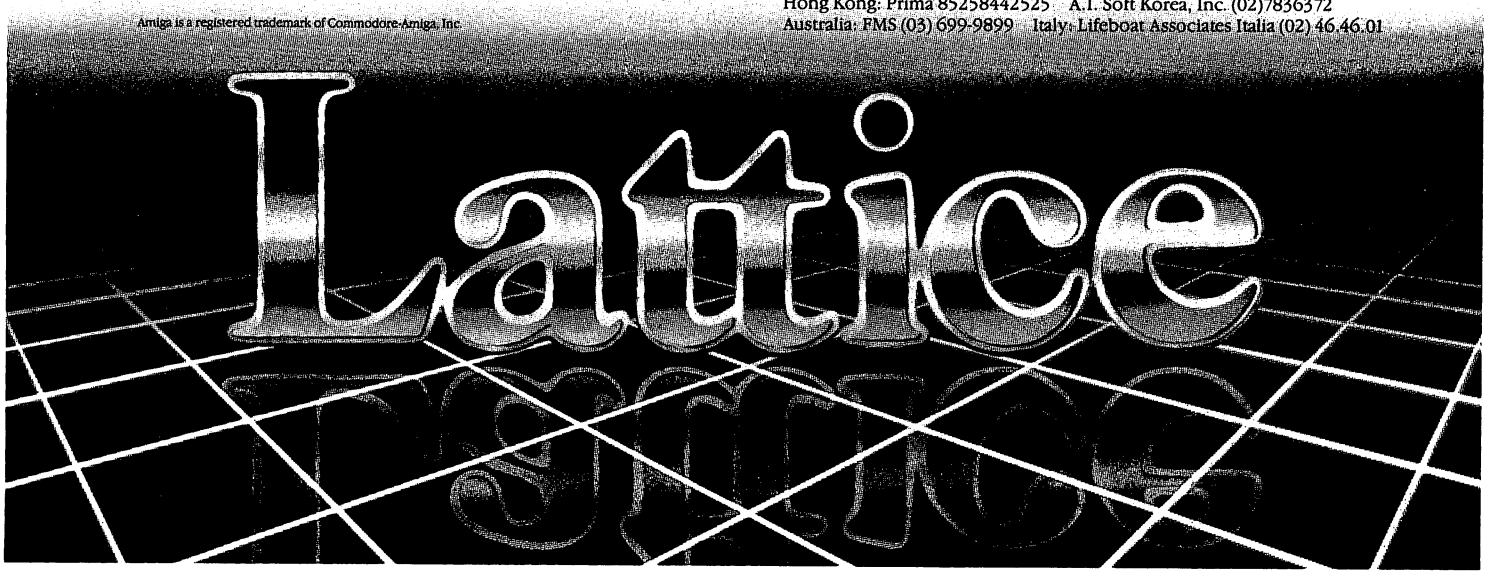
Panel will help you write your screen programs and layer your screen designs with up to ten overlapping images. *Panel's* screen layouts can be assigned to individual windows and may be dynamically loaded from files or compiled into a program. *Panel* will output C source for including in your applications. A monitor and keyboard utility is also included to allow you to customize your applications for other systems.

With Lattice products you get *Lattice Service* including telephone support, notice of new products and enhancements and a 30-day money-back guarantee. Corporate license agreements available.

Lattice

Lattice, Incorporated
Post Office Box 3072
Glen Ellyn, Illinois 60138
(800)533-3577 In Illinois (312) 858-7950
TELEX 532253 FAX (312) 858-8473

INTERNATIONAL SALES OFFICES: Benelux: Ines Datacom (32)2-720-51-61
Japan: Lifeboat, Inc. (03)293-4711 England: Roundhill (0672)54675
France: Echosoft (1)4824.54.04 Germany: Pfotenhaur (49)7841/5058
Hong Kong: Prima 85258442525 A.I. Soft Korea, Inc. (02)7836372
Australia: FMS (03) 699-9899 Italy: Lifeboat Associates Italia (02) 46.46.01



Amazing Mail:

Greetings from Ireland,
I would like to say that each edition of your excellent publication is more interesting than the one before and I hope that this will continue.

I have taken this opportunity to include some information about INFORMATIQUE, an AMIGA based online information service, and maybe if you have space you might mention INFORMATIQUE in a future edition, you might also mention that I would like to contact others operating Amiga-based Bulliton Boards.

As I am using SIDECAR since Christmas I was suprised to get the impression from reading the most recent edition of AMAZING that SIDECAR is not readily available in the USA. I am fairly pleased with SIDECAR but getting answers to technical questions is like pulling teeth (painful).

In the manual supplied with SIDECAR an internal AMIGA RAM expansion is mentioned (0.5 MEG or 1 MEG) as being available but no one within Commodore, at least on this side of the Atlantic, seems to know where I can get suitable expansion units for the SIDECAR (it is interesting to note that various newsletters issued to developers mentioned RAM expansion of as much as 2 MEG). On page 13 of your most recent issue edition(Vol 2,#3) I noticed the following: " This is the same memory upgrade card that fits into sidecar" so I would like to ask this question .... Has anyone ever seen this board and is it available to end users?

My second problem is that I was led to believe that file transfer facilities between AMIGADOS and MS-DOS would be supplied with SIDECAR... this facility is very important for the operation of my BBS but on getting my unit I discovered that this was not possible and at a later date discovered that the required software was not available and up until this Friday Commodore (UK) had no idea as to when it might be available and yet it would appear that GAIL WELLINGTON could impress the Press at CES by transferring text between MS-DOS and AMIGADOS windows.... this sort of thing really does annoy me and it is holding back my expansion plans, I am afraid to purchase a Hard Disk in case I discover that suitable software is not really available outside Trade Shows. If you can obtain answers regarding the above please let me know .... I wrote to Germany and they did not reply and Commodore in the UK seems to know less than I do about the subject.

Before I say goodbye it should be mentioned that I am a dedicated Commodore user and as I am convinced that they do make the best equipment (maybe by accident) it is my intention to continue using their products.

Regards
William John Murphy
Ireland

In the US introduction of the AmigaA500 and Amiga A2000 presentation given at the Boston Computer Society 3/25/87, references were made to both the Ram expansion and the file transfer abilities (beyond those available through 1.2) for the A2000. The agreement was that both would be ready by the time the A2000 is available in the US in May 1987 (?).

Dear Amazing ,
Well, here I am again in the Commodore Underground. I feel I ought to be wearing a beret and smoking short yellow cigarettes. I had a C64 when they first came out , back when the only software was what you wrote . Of course , now the C64 market is wide open. Now I've jumped into an Amiga at the same bleak stage.

Nothing erodes a new user's confidence in a computer more quickly than the impression that he's bought an orphan. Despite an avalanche of assurances that Amiga software and support is under intense development, it's still very hard to find in the Las Vegas area, and from the letters I read in your magazine and others, the situation is not much better elsewhere. The only stores that deal with the Amiga are chiefly dealers for other systems who have taken on the Amiga as a sideline. Therefore I think you may appreciate the following:

In your Amazing Dealers page of your V2.2 issue, I noticed a listing for Ridgecrest Computer Center in Ridgecrest, CA. Since I was about to take a trip to that area, I called them from Vegas to see if they had a modem in stock. They did, and assured me that they would hold it for me to pick up the next day. I asked for terminal software and they had none, but said they would check for public domain software with the local Amiga Users Group. Fifteen minutes later they called me back, LONG DISTANCE, to tell me the AUG president would drop off Communications 1.33 for me, not to worry!

When I got to the store, I noticed their main product line is Atari, and they told me they were just taking the Amiga line on board. Their service was as good as their word, and everyone who worked there was friendly and helpful, and knew what they were talking about. A refreshing change from the local dealer who told me that as far as he knew, only the Amiga Modem would work with the Amiga, and he didn't have one in stock.

The bottom line is product support. Dealers who follow that line will get my business and my recommendation to other users. Since I know too well that Commodore's history of support for their own products is - Well, erractic at best, I fully appreciate dealers like this who take that "extra step" in their stride. Maybe "Big" dealers can't afford to do it....so I'll keep dealing with the people who can.

Thanks for your time, and thanks for a fine magazine - if it weren't for your articles and columns, I'd still be wallowing around trying to figure out how to talk to the Amiga.

P.S. Lead on Borland for AmigaTurbo, will you? Mod-2 is all very well, but I really don't need to learn yet another language.

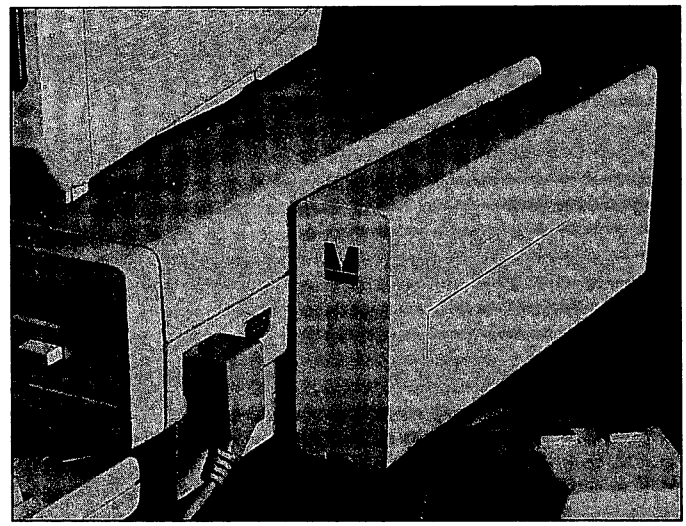
Alan W. Thompson
442 Burton St.
Henderson, NV 89015

We hope that both our Amazing Dealers and Advertisers will maintain a high level of service. However, if you do have a problem, please send AC a written complaint and we will forward your concerns. •AC•

AVAILABLE NOW!

StarBoard2

Auto-Configuring Fast RAM
Zero Wait States
User Expandable from 512k to 2 Megabytes
Bus Pass-Through
MultiFunction Option: battery/ clock, FPU, parity, Sticky-Disk



If you've owned your Amiga® for a while now, you *know* you definitely need more than 512k of memory. You probably need *at least* double that amount...but you might need as much as an additional two megabytes. We want to urge you to use **StarBoard2** as the solution to your memory expansion problem –and to some of your other Amiga-expansion needs as well!

It's small, but it's BIG–

Since most of you want to expand your Amiga's memory without having to also expand your computer table, we designed **StarBoard2** and its two optional "daughterboards" to fit into a sleek, unobtrusive Amiga-styled case that snugly fastens to your computer with two precision-machined jackscrews.

The sculpted steel case of **StarBoard2** measures only 1.6" wide by 4.3" high by 10.2" long. You can access the inside of the case by removing just two small screws on the bottom and pulling it apart. We make **StarBoard2** easy to get into so that you or your dealer can expand it by installing up to one megabyte of RAM on the standard **StarBoard2** or up to two megabytes by adding in an Upper Deck.

This card has decks!

The basic **StarBoard2** starts out as a one megabyte memory space with 0k, 512k, or one megabyte installed. If you add in an optional **Upper Deck** (which plugs onto the Main Board inside the case) you bring **StarBoard2** up to its full two megabyte potential. You can buy your **StarBoard2** with the Upper Deck (populated or unpopulated) or buy the Upper Deck later as your need for memory grows.

And you can add other functions to **StarBoard2** by plugging in its second optional deck –the Multifunction Module!

StarBoard2: functions five!

If we count Fast Memory as one function, the addition of the **MultiFunction Module** brings the total up to five!

THE CLOCK FUNCTION:

Whenever you boot your Amiga you have to tell it what time it is! Add a MultiFunction Module to your **StarBoard2** and you can hand that tedious task to the battery-backed, real-time clock/calendar. A small piece of MicroBotics software in your WorkBench Startup-Sequence reads the clock and automatically sets the time and date in your Amiga. And the battery *is* included (we designed it to use an inexpensive, standard AAA battery which will last at least two years before needing replacement).

THE FLOATING POINT FUNCTION:

If any one aspect most characterizes the Amiga it's *fast* graphics! Most graphic routines make heavy use of the Amiga Floating Point Library. Replacing this library with the one we give you with your MultiFunction Module and installing a separately purchased Motorola 68881 FPU chip in the socket provided by the Module will speed up these math operations from 5 to 40 times! And if you write your own software, you can directly address this chip for increased speed in integer arithmetic operations in addition to floating point math.

THE PARITY CHECKING FUNCTION:

If you install an additional ninth RAM chip for every eight in your **StarBoard2**, then you can enable *parity checking*. Parity checking will alert you (with a bus-error message) in the event of any data corruption in **StarBoard2**'s memory space. So what good is it to know that your data's messed up if the hardware can't fix it for you? It will warn you against saving that data to disk and possibly destroying your database or your massive spreadsheet. The more memory you have in your system the more likely it is, statistically, that random errors will occur. Parity checking gives you some protection from this threat to your data residing in Fast RAM. Note that the Amiga's "chip" RAM cannot be parity checked.

THE IMMORTAL MEMORY DISK FUNCTION (STICKY-DISK):

When you've got a lot of RAM, you can make nice big RAM-Disks and speed up your Amiga's operations a lot! But there's one bad thing about RAM-Disks: they go away when you re-boot your machine. Sticky-Disk solves that problem for you. It turns all of the memory space inside a single **StarBoard2** into a Memory Disk that will survive a warm-reboot! When your Amiga attempts to grab a **StarBoard2** in Sticky-Disk mode, a hardware signal prevents the system from acquiring the **StarBoard2** as FastRAM (and thereby erasing your files) –instead it is re-recognized as a Memory Disk and its contents are preserved intact. If you want to work rapidly with large files of data that are being constantly updated (such as when developing software) you can appreciate the Sticky-Disk!

Fast RAM –no waiting!

StarBoard2 is a *totally* engineered product. It is a ZERO WAIT-STATE design, auto-configuring under AmigaDOS 1.2 as Fast RAM. Since AmigaDOS 1.1 doesn't support autoconfiguration, we also give you the software to configure memory in 1.1.

Any applications software which "looks" for Fast RAM will "find" **StarBoard2.** And you'll find that your applications run more efficiently due to **StarBoard2** on the bus.

A passing bus? Indeed!

What good is an Expansion Bus if it hits a dead end, as with some memory cards? Not much, we think –that's why we carefully and compatibly passed through the bus so you could attach other devices onto your Amiga (including another **StarBoard2**, of course!).

The sum of the parts...

A really nice feature of the **StarBoard2** system is that you can buy exactly what you need now without closing off your options for future exapansion. You can even buy a 0k **StarBoard2** (with a one megabyte capacity) and populate it with your own RAM (commonly available 256k by 1 by 150ns memory chips). When you add **StarBoard2** to your Amiga you have a powerful hardware combination, superior to any single-user micro on the market. See your Authorized Amiga Dealer today and ask for **StarBoard2**

SUGGESTED RETAIL PRICING:	
StarBoard2, 0k (1 meg space):	$349
StarBoard2, 0k (2 meg space):	$395
StarBoard2, 512k (1 meg space):	$495
StarBoard2, 1 meg (1 meg space)	$595
StarBoard2, 2 megs installed:	$879
StarBoard2, 2 megs & MultiFunction:	$959
Upper Deck, 0k (1 meg space):	$ 99
MultiFunction Module:	$ 99
also available:	
Standard 256k memory card:	$129
MAS-Drive20, 20 meg harddisk:	$1495
MouseTime, mouseport clock:	$ 50

MicroBotics, Inc.
811 Alpha Drive, Suite 335, Richardson, Texas 75081 / (214) 437-5330

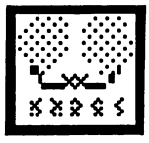
Amazing Amiga Digitized Sound!

AC Reviews

FutureSound™
PerfectSound™
SoundScape™ Sound Sampler

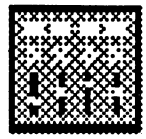
SoundScape... Power Play for the AMIGA.

SoundScape Pro MIDI Studio



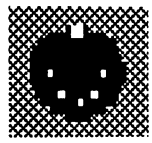
The most powerful performance and recording software on any computer. The recording studio-like environment provides complete facilities for routing, recording, editing, transposition and playback of any musical performance. As new modules are introduced, you can "install" them at any time. Music can be performed by the internal sampled sound synthesizer, or with any external MIDI equipment. Record from the QWERTY keyboard or any external MIDI source, including keyboards, guitar and pitch followers. Synchronize with, or provide MIDI clock information, including MIDI Song Pointers. The complete flexibility of the system makes your imagination the only limit to its power.

- Number of notes and tracks determined by available memory
- MIDI patch panel links program modules
- Install new modules at any time
- Up to 16 internal instruments at one time
- Complete sample system with editing, looping, ADSR envelopes, velocity sensitivity, and pitchbend.



- Up to 160 sampled sounds at one time
- Save and load IFF note and sample files
- Quantize to any multiple of MIDI clock beats
- "Match" mode eases learning of a song
- Complete MIDI sequence and song editing
- Route, merge, split, or bounce any track to any other.

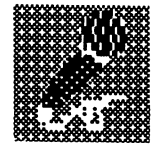
SoundScape MIDI Interface



Necessary for any program which supports MIDI to communicate with MIDI equipment.

- Completely compatible with the standard Amiga MIDI interface
- MIDI In, Out, and Thru connectors
- Plugs into the serial port

SoundScape Sound Digitizer



With the SoundScape Sound Digitizer, any sound may be sampled and modified by the Amiga, including voice. IFF File compatibility enables these samples to be used as musical instruments, sound effects, or speech with any IFF compatible music or animation system.

- High quality
- Highest possible fidelity from the Amiga
- Stereo or mono
- Variable sample rates
- Mike and line inputs
- Digitally controlled volume on each channel
- IFF Sample File compatible
- Software included for sampling, editing, and MIDI performance functions

Available From Your AMIGA Dealer.

SoundScape Pro MIDI Studio	**$149.00**
AMIGA MIDI Interface	**$ 49.00**
SoundScape Audio Digitizer	**$ 99.00**

Amiga is a trade mark of Commodore Business Machines

Prices and availability subject to change without notice

mimetics™

corporation **...the professional software source!!**

P.O. Box 60238 Sta. A, Palo Alto, CA 94306 (408) 741-0117

Amazing Reviews...

SoundScape

Pro MIDI Studio

A powerful music editor / player

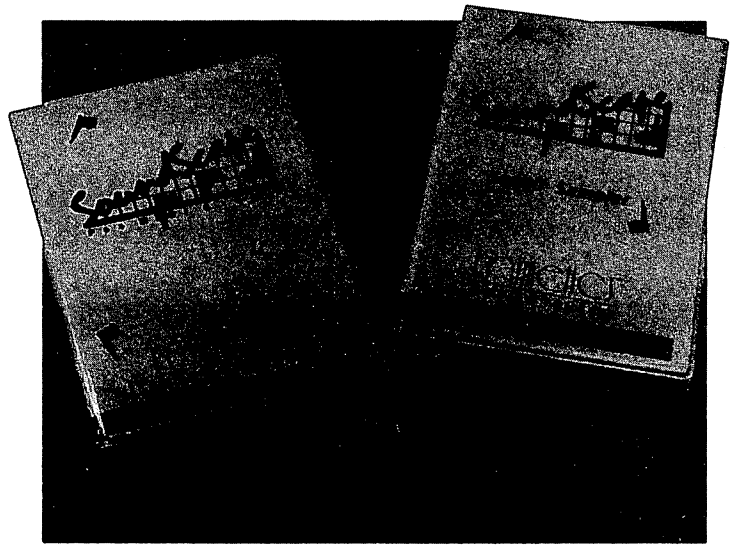
Reviewed by
Jeffrey Sullivan

When Mimetics released their SoundScape Pro MIDI Studio, the champions of the Amiga as the" ultimate music machine" breathed a heavy sigh of relief. At last, a program for the serious musician. On the other hand, there is the new breed of software for the Amiga and its sister 68000 machines which is supposed to allow the user to interface with the computer with a minimum of technical fuss. At which end of the spectrum does this program lie?

As you may have guessed from the buildup, SoundScape falls about in the middle of the range. It is both fairly powerful and fairly easy to use. Mimetics has put together a useful package for both the experienced musician and the musical novice.

SoundScape is touted as "the most powerful performance and recording software on any computer". This is a very powerful claim, but one which Mimetics has made a credible attempt at fulfilling. SoundScape is not actually a program, but rather a collection of modules tied together with the SoundScape Music Operating System. This operating system is a full multitasking environment, transparent to the user, which allows for the transfer of musical data between the different modules of the system. Although Mimetics claims that SoundScape will run concurrently with other programs, this assurance must be taken with a grain of salt. While it is true that you can run other software with SoundScape given sufficient memory, 512K is not enough for you to do so. I tried to run SoundScape along with Scribble! while I was writing this review. Every time I tried, I got a meeting with the guru. However, if you have extra ram or are running small programs, SoundScape does support multitasking.

Booting up SoundScape will bring up a window called the Patch Panel. This window is analogous to the patch panels used by musicians to connect all their equipment. On the left side of the window are the possible input modules: the MIDI mixer, console keyboard, MIDI In, and clock modules. On the right side of the window are the output modules: the sampled sound player, MIDI mixer, player piano, tape deck, MIDI Out, and clock modules. Some modules appear on both sides of the Patch Panel because they can serve both as input and output modules. For example, the MIDI mixer can supply output to the tape deck, and in turn, receive input from the MIDI In module.



To connect any of the input modules to the output modules, you simply click on the icon for the input module, then click on the icon for the output module. A given input module may be connected to many output modules and vice versa. When you connect two modules, a window may or may not open up which contains controls for that function or more information about it. A quick overview of each module will help to explain its purpose within the program. Basically, each module serves to receive input from some source and to transform it in some way before shipping it to another module.

The input sources in SoundScape are the MIDI mixer, console keyboard, MIDI In, and clock modules.

continued...

The MIDI mixer simply accepts input on any of the 16 MIDI channels, mixes the data together and sends the input out on the given tracks. This process allows you to combine, say, input from a MIDI synthesizer and input from the console keyboard and then send the information to the tape deck on a single channel.

The console keyboard allows the user to enter musical notes into the system by playing the top row of the Amiga keyboard (QWERTYUIOP{}, with number keys used for flats and sharps) as if it were a musical keyboard. The console keyboard can also send some MIDI timing information for use with certain recording modes. The console keyboard window is a section of a piano-type keyboard with the appropriate letters beneath each key. Each key flashes when depressed.

The MIDI In module allows the user to connect any MIDI equipped device (e.g., synthesizers, drum machines, etc.) through a MIDI interface and send musical notes from the MIDI device.

The clock sends timing signals to the other modules and MIDI devices connected to the system. The timing signals are needed to coordinate the recording and playback functions of SoundScape. The clock window's controls are much like a cassette deck's controls (stop, play, play from beginning, fast forward, reverse). There are also some less familiar functions (clock lock, and clock on with go). There are also two auto-locate registers which allow you to save the counter value at a particular place in a song, simply by clicking on a button. This option is very handy when you are tweaking a piece of music into shape. The clock's counter will increment by MIDI clocksallower second or by beats per minute, depending on user preference. The clock module also has a slider to let you to adjust the tempo of the piece.

The output modules in SoundScape are the sampled sound player, MIDI mixer, player piano, tape deck, MIDI Out, and clock modules.

The sampled sound player plays back musical scores through the Amiga's four sound channels using a variety of sampled instrument sounds. Instruments are loaded into any one of 16 slots. Each slot corresponds to a MIDI channel and any output sent to that channel will be played by that particular instrument. If you own a Mimetics Sound Sampler (or equivalent sounds sampler), the sampled sound player also allows you to create your own sampled instruments. The sampling window allows you to translate octaves of the instrument, set the instrument's responsivity to velocity and pitch modulation, tune the instrument and set a transpose value to replay the notes up to a full octave in half-step increments.

You can load and save instruments individually or in sets, called instrument banks. This feature is useful if you have finished a score and know exactly which instrument you want for each part. With instrument banks , you can load them all with one command.

The player piano is a window containing a four-octave keyboard which can show the notes played from any two MIDI channels. One channel's notes are played in red, the other channel's in grey.

The tape deck is the workhorse of SoundScape. The tape deck both records and plays back musical information on any number of tracks. For those familiar with such terms, the tape deck is a MIDI sequencer. The deck is modeled after a multi-track tape deck, like those used in recording studios for ease of use. Each track in the tape deck can be given a different input and output for maximum flexibility and power. The tape deck has controls similar to those on the clock, plus an additional set of commands to load, save, create, edit, and manipulate tracks.

The MIDI Out module sends the musical data out through the MIDI interface to a MIDI equipped device such as a synthesizer or drum machine for performance.

Playing Songs

SoundScape allows you a number of playback options. The two most common options are experimenting with the console keyboard , and playing back scores with the tape deck.

Whichever method you use, you can send your music to the sampled sound player, the MIDI Out module, or both. I commonly experiment using the sampled sound player and then send the final pieces to my synthesizer.

To experiment with the console keyboard, simply connect it to the sampled sound player and load in the instruments you want to work with. You can play only one instrument at a time, since the console keyboard only sends data on one channel. You can, though, tell the keyboard which MIDI channel to send on to choose your instrument. The notes you play will flash in the console keyboard window and the music will come out of your Amiga's sound port (if you want to really enjoy it, hook up to a stereo-- the sound difference is amazing!).

To play back completed songs, hook up the tape deck to the output module of your choice and load in a score. By the way, all of the load requesters in SoundScape load the directory once and keep it in memory to minimize the time spent waiting for the them to come up. If you change directories or save something to the directory, simply select the directory gadget and either change it to the directory you want, or just hit return to re-read the current directory. This feature is very much appreciated if you've ever spent a lot of time in DPaint loading and unloading pictures.

Once a score is loaded, you can use either the clock controls or a similar set of controls on the tape deck to play, stop, fast forward or rewind your song. You set the output module for each track of your score with iconic gadgets in the tape deck. Each track can be given a different output destination if so desired. Perhaps you would send the first four channels to the sampled sound player and the other channels (up to 16) through the MIDI Out module to your synthesizer (if it is multi-timbral).

Recording Scores

While playing back is the ultimate goal of composing music, recording is the intermediate step. SoundScape gives you excellent tools for use in perfecting your score. You can record your performance in real-time (live) or you can use step- time recording to record single notes.

Recording in real time simply causes SoundScape to record all of the notes which you play, their durations, and other musical data if your input device can send it (such as key velocity and pitch bend).

In step- time recording, you press the key of the note you want to enter, then use the Amiga function keys to send a certain number of MIDI clocks to the tape deck(sequencer). This process tells the tape deck how long the note is supposed to be. This mode is great for those of us who are not great musicians, or those great musicians who want to record impossible riffs.

SoundScape also supports a punch in/punch out recording mode. This mode allows you to play back your song to a point where you want to change it, then record new information over the part you want to change. The tape deck uses a punch in and a punch out register to accomplish this task. It plays your song back until the counter reaches the value in the punch in register, then records until the counter reaches the value in the punch out register. This is extremely handy for repairing parts of your score without having to edit every note individually.

The tape deck lets you filter MIDI events as well, such as pitch bend, key velocity, aftertouch pressure, etc. These events can take up much memory; a few seconds of pitch wheel use can generate hundreds of MIDI events! Also, people without these capabilities in their synthesizers would want to filter this information out.

The capacity of the tape deck's sequencer is theoretically infinite---infinite in the sense that you are limited only by the amount of memory available. The amount of free memory you have available depends on what instruments you have loaded in the sampled sound player (the piano samples are 98K), and whether you are running any other programs simultaneously.

Whenever you are either playing or recording, it is possible to save every setting and parameter as an Environment. This environment is all the patch panel settings, all loaded instruments, and scores. It is ideal to be able to save everything as an environment and call it up with a single command.

continued...

Editing Scores

Once you have entered your score, you will surely want to edit it. SoundScape has a MIDI editor which allows you to edit any MIDI event and change any of its parameters. You can change the duration of a note, change which key was pressed, etc. You can also use an area called the list store to hold segments of musical data which you may want to use in other tracks or in other parts of the current track. The editor allows you to splice and loop notes and sequences, as well as quantize sequences to any multiple of MIDI clocks.

In addition to the MIDI editor, there is a song editor which works on the larger picture, combining sequences to form songs. You can splice sequences and transpose them to a different key up to a full octave in half-step increments.

Both editors are full implementations and give you total control over your music. They are well documented and very easy to use.

Special Modes

In addition to the standard recording and playback modes, SoundScape includes a few special modes which allow some pretty neat things to be done.

Echo mode waits for the user to press a key. The tape deck then plays back a pre-recorded sequence, but transposes it by the amount of difference between the key and middle C. This feature allows you to play back a repetitive riff, but to vary its pitch.

Trigger mode is similar to echo mode in that it waits for the user to press a key, then plays back a pre-recorded sequence. The difference is that in trigger mode, the key pressed is a 'trigger' for a specific sequence. No transposition takes place and many different sequences may be set up, each triggered by a different key. This mode is tremendously useful. A performer could record a number of different parts, setting each to be triggered by a specific key. When the performer wants to play that specific sequence, he simply hits the appropriate key and SoundScape plays back the error-free sequence.

Match mode is another useful mode. In match mode, the tape deck plays a note in a sequence, then waits for that note to be entered on the keyboard. The deck then plays the next note and waits, and so on. This feature is a good way to learn a certain piece; just enter the piece into the tape deck (using step time recording if need be), and then set the tape deck to match mode. The tape deck will wait for you to correctly echo the given note before playing the next note.

Gripes

While SoundScape is an excellent music program, it is not without its problems. First and foremost, you may say to yourself "Sure, this system is great if you have a synthesizer, but I just have my Amiga." This is a legitimate point. SoundScape is definitely better if you have a synthesizer. However,SoundScape is also useful with just your Amiga. The sampled sound player is an excellent source of high quality instrumental sounds and should not to be sold short. It is possible to do some things with SoundScape that cannot be done with any other music package. Of course, SoundScape also has a utility for transferring music between Mimetics and IFF formats. Additionally, a decent synthesizer need not be expensive; take for example the Casio CZ series which has units as inexpensive as $250, or the Yamaha DZ-100, which is another excellent bargain. If you are at all serious about computer music, I strongly recommend that you consider purchasing a synthesizer.

The second major problem with the Soundscape system is that it has no way of displaying music in standard notation (notes on a staff). Musical notation is in MIDI events such as Note On, Control Change, Pitch Wheel, and Patch Change. I spoke to Mimetics about this problem, and they informed me that they already had a standard notation module completed, but were waiting to see what other developers were bringing out.

So one way or the other, probably by the time you read this, there will be standard notation available for SoundScape. Also, you could transfer the score to IFF format and view it in any program with standard notation that can read IFF files.

Another liability of SoundScape is a result of its fantastic power. SoundScape is not as easy to use as some other music programs. Such difficulty is to be expected in a program of this sophistication. However, SoundScape is still fairly easy to use, and given its complexity, the credit goes to programmer Todor Fay and the folks at Mimetics who have made it this easy to use.

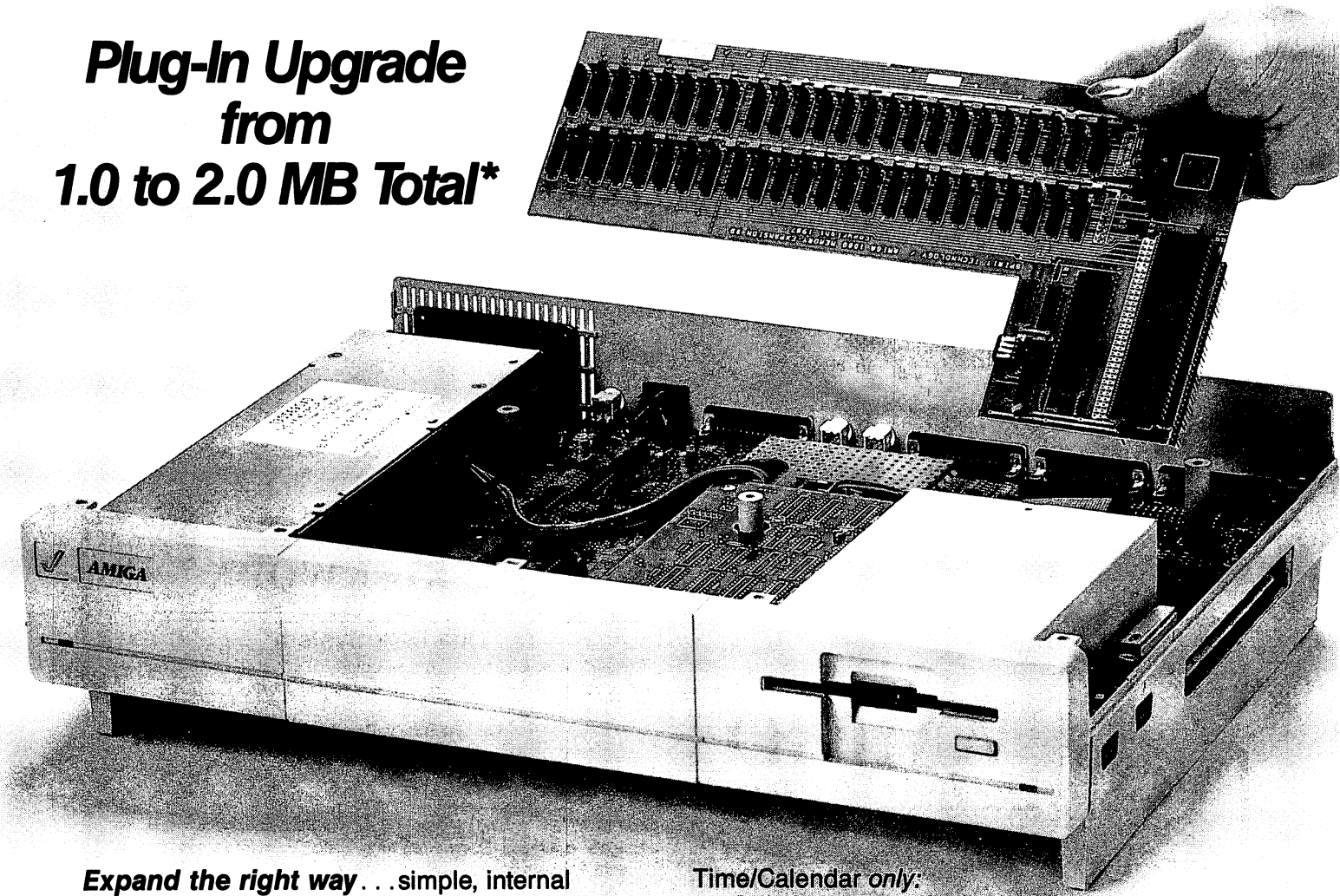
As with any new skill, it is good to have a lot of examples to consult when you are trying to figure something out, or to demonstrate some neat ways to exploit the power of the system. SoundScape was a bit remiss in this area; the package comes with only four scores and a metronome. This lack of scores can be mediated by the use of the IFF conversion utility. It is obvious why so few examples were included. The SoundScape disk is almost totally filled by the syst,em itself, leaving little room for demos. Still, I think that they could have included a second disk of scores, considering the price of the package.

The documentation is generally very good, but there are certain deficiencies which bear mentioning. The score conversion program was not noted at all in the manual. The only way to determine this program's use was to run the module by double-clicking on it (it does not seem to be directly callable from SoundScape). This deficiency may be a result of the recent release of the product, however.

SoundScape also lacks instrument sounds. The people at Mimetics have informed me that they will be marketing a disk of presampled sounds soon (probably available by the time you read this). Any standard IFF sampled sound file will work, but unless you own another music program or can download sample files (and they're BIG!) you may have trouble. Mimetics does sell a sampler which you can use to make your own sounds as well.

AMIGA INTERNAL MEMORY EXPANSION
Plus
TIME/CALENDAR

Plug-In Upgrade from 1.0 to 2.0 MB Total*



Expand the right way. . . simple, internal plug-in mounting leaves your side expansion port free to add other peripherals. Also, the internal Time/Calendar does not use a joystick port.

Memory Expansion Features:

* Zero Wait-State
* No Cuts or Soldering Required
* Full Auto-Configuration
* Lithium Battery Back-Up for Time/Calendar

ORDERING INFORMATION:

DRAM Memory *with* Time/Calendar:

#ST-05	0.5 MB	$349.50 List
#ST-10	1.0 MB	$499.50 List
#ST-15	1.5 MB	$599.50 List

**Memory expansion from 1.0 to 2.0 MB includes AMIGA 1000 512K RAM.*

Time/Calendar *only*:
#ST-TC Time/Calendar including Battery Back-Up $59.50 List

ASK ABOUT increased speed with the *new* 68010 Processor

VISA and Mastercard Welcome

CALL TOLL FREE: 1-800-433-7572

Factory direct: 1-801-485-4233
DEALER INQUIRIES INVITED

SPIRIT™ TECHNOLOGY

220 West 2950 South • Salt Lake City, Utah 84115

There is also a bit of a problem with the manner in which SoundScape searches for a given file type (score, instrument, etc.). The load requesters are fast and easy to use, but they search the disk for files using a certain naming convention. For a sampled sound, the file name must end in '.samples' or SoundScape will not recognize it and will, therefore, not include it in the load requester. I have transferred all of my Instant Music samples to a SoundScape workdisk, but I was forced to rename every one of the files to end in '.samples' which was annoying. I would be nice if you could specify the search pattern in the requesters, so that it would not be necessary to rename files just to read them.

Also, some of the windows in SoundScape can not be open simultaneously. I don't know why this is, but it must have involved some poor planning by Mimetics. For example, the console keyboard control window and the sampled sound player window cannot be open at the same time. This means that if you are searching for a certain sound or just playing with the sounds, you have to load all of the sounds you want and write their channel numbers down. Otherwise, you have to switch back and forth between the two windows, closing one and opening the other each time you want to change the sound you are playing. This quickly gets tiring.

Summary

To summaize, SoundScape by Mimetics is an excellent music editor/player for the serious music enthusiast. It has a few flaws, but overall makes a recording software on any computer." Mimetics has also informed me that they are developing additional modules that will interface with SoundScape. Some possible topics to be covered are standard musical notation and keyboard tutorial/instruction. A developer's tool kit is also available to those people interested in writing their own SoundScape modules and utilities.

SoundScape Pro MIDI Studio $149.00

Mimetics Corporation
PO Box 60238 Sta. A
Palo Alto, CA 94306
(408) 741-0117

•AC•

Amazing Reviews...

The PERFECT SOUND Digitizer

by Ron Battle

When you purchased your Amiga, you probably thought about buying some interesting hardware --- a digitizer, perhaps. For some time now, a wide array of sound digitizers has been available for the Amiga. Unfortunately, the inexpensive models had limited capabilities and the full featured digitizers weren't affordable! Until now. Enter the PERFECT SOUND digitizer from SunRize Industries, a full-featured stereo digitizer that retails for only $79.95.

HARDWARE FEATURES...

The Perfect Sound digitizer connects onto the back parallel printer port. Extending from the front of the digitizer are two knobs which control the input amplifier gain for the left and right channels. Unlike other digitizers, PERFECT SOUND will record in true stereo or separate left and right channels. You cannot hook a microphone directly to the RCA phono plugs because there is no pre-amplifier. The built in 8 bit A/D converter can sample at a variable rate, from 5,000 to 25,000 times per second. If you compare other digitizers to PERFECT SOUND, you will note that many have a fixed sampling rate and/or cannot sample in STEREO. There are no hardware provisions to change the input volume under software control; this must be done manually by twisting the knobs.

SOFTWARE FEATURES...

The sound editor included in the price of the digitizer is actually a public domain program and includes the full source code in C ! The program is intuition- based with pull down menus for editing and special features such as graphing waveforms, digitizing sounds, and filing. The slider gadgets are used for extracting portions of a waveform to be deleted, inserted in other sounds, or filed as a new sound. You have full control of the recording period, playback period, which channel(s) to digitize. You can also flip a sound "backwards" and and monitor sound levels of input signals. You can save and load your sound files three ways: IFF format, a DUMP format that saves only digitized info, or COMP format that uses Fibonacci Delta Compression techniques to cut your file size in half, with a degradation of sound quality. Also on this disk is a program to monitor sounds using interrupts.

You also have access to numerous digitized sounds including a bouncing ball, crowd sounds, helicopter, and more. There is even a file with documentation on the IFF format. The manual is 19 pages long, short and concise, but doesn't fully explain how to create new instruments for INSTANT MUSIC.

HOW TO CREATE NEW INSTRUMENTS FOR INSTANT MUSIC...

I will create a new instrument called LA which will actually be my voice saying "la". First of all, my hardware setup includes a microphone hooked to my cassette tape recorder. I then have a cord connecting the OUT port of the tape recorder to the LEFT (bottom) RCA port on the sound digitizer. Now, when I want to record my voice, I start the tape recorder, and while in the Perfect Sound Editor, I select the RECORD A SAMPLE and LEFT channel from the menu.

The program then waits for me to press the LEFT mouse button. I press the button and say "la" into the microphone and then press the LEFT mouse button again to stop digitizing. Now, my voice is captured for posterity! I can then use the editor to graph out my voice and cut away noise from the beginning and end of the sound. This editing is

continued...

accomplished by moving the START and END sliders on the screen until I get clean sound . I then COPY RANGE TO NEW SLOT and name this sound "LA". Be sure click on the old sound and DELETE it before proceeding to make a new instrument!

Now select FREQ=FREQ/2 from theSPECIAL menu to make a new "LA" sound that is half the frequency of the orginal. You can also click the pointer on the original "LA" sound and then select FREQ= FREQ*2 to make a new "LA" sound that is twice the original frequency. You will now have three sounds on the screen: the original "LA", half this frequency, and twice the original frequency. Now, click the pointer on the original sound, move the START slider all the way to the right (since there is no repeating part to your new instrument) and select CREATE INSTRUMENT from the SPECIAL menu . You now have a three octave instrument based on your own voice! Save this instrument as LA to your RAM: device.

The tricky part now is to get this creation onto the INSTANT MUSIC disk! Unfortunately, the original INSTANT MUSIC disk does not have any room for new instruments. Thus, you must make a copy of the original and delete some instruments from this copy. This process is a little bit tricky because the directories of instruments have a SPACE in their names such as LIBRARY 1, LIBRARY 2, LIBRARY 3. If your copy of Instant Music is in DF1:, try the following: DELETE "DF1:instruments/library 1/sax". You MUST use the quotes because of that space in the directory name. It took me a while to figure tha one out!! Now, type COPY RAM:LA "DF1:instruments/library 1". You have now copied your new instrument LA to the directory INSTRUMENTS, subdirectory LIBRARY 1. When you boot up the Instant Music disk copy, you now have LA as a new instrument!

I have assumed throughout this discussion that you are familiar with the command line interface (CLI) and other AmigaDOS commands. It is much easier to get new sounds into DELUXE VIDEO. If your new sound is called MYSOUND and located in the RAM: device, and your copy of DELUXE VIDEO (Maker disk) is in device DF1:, then type: COPY RAM:MYSOUND DF1:SOUNDS . That's it! You can also include INSTANT MUSIC songs that use your new instruments, in videos made with DELUXE VIDEO. You just have to be sure to use the Options menu in Deluxe Video and set up the data drawers to point to your new instruments. In this example, if a song used LA, you would set up the instruments drawer as DF1:instruments/library 1. Make sure the rest of your instruments are in that drawer! Both of the Electronic Arts programs, DELUXE VIDEO and INSTANT MUSIC, seem to work best when your sounds or new instruments are less than 24K bytes long.

WHAT I DON'T LIKE...

I have an old version of the PerfectSound editor (v1.93) that gives me 216,800 bytes of RAM to record on. Unfortunately, with the latest version (v2.0), there are only 72,592 bytes of RAM to record sounds after all the demos are loaded! You don't get any more space by deleting the demos (or I haven't figured it out yet!). If you change your disk so that the demos don't load, you then get 185,848 bytes of RAM to play with! Since SunRize Industries is constantly updating their software, I would guess that this problem may be already solved by the time you read this! In terms of hardware, it would be nice to have a pre-amplifier port on board with low-pass filter built in, so a microphone could be directly hooked to the digitizer. Another feature on my wish list would be the ability to change the input volume under software control while recording sounds and the option to change the volume under software control during playback. These features would be helpful when doing a technique called companding; a sneaky way to get 12 bit resolution on an 8 bit digitizer!

WHAT I LIKE...

This digitizer is fun and quite addictive! I really like having the source code, so I can figure out how things were done and so I can modify the program to fit my needs. When you consider the hardware and software features of this sound digitizer, the price is very reasonable; I highly recommend the PERFECT SOUND digitizer!

WHERE TO GET ONE...

SunRize Industries
P.O. Box 1453
College Station, Texas 77841
(409) 846-1311

The digitizer comes with a 90- day warranty and there is also a technical support phone number you can call if you have any questions. The retail price is $79.95.

ADDENDUM...

SunRize has just released version 2.1 of Psound. In this version the original bugs have been fixed with memory allocation and the general performance has been improved . SunRize has also released Studio Magic, an advanced editor capable of doing echoes, frequency enhancements, FFTs, and more! List price is $69.95.

•AC•

Amazing Reviews...

The FutureSound Sound Digitizer

By Warren Block

The Amiga's technique for making sound is relatively simple. A sound is digitally recorded or *sampled*, to be played back later at different pitches and rates. This approach has the advantage of creating high-quality, realistic sounds, but requires a combination of specialized hardware and software to record the sound in the first place. ***FutureSound*** provides a comprehensive solution to this bulky problem.

Hardware

FutureSound's hardware consists of a small (5 x 5.25 x 1.5-inch) box with a cable which plugs into the Amiga's parallel printer port. After connecting the cable to the computer, the printer cable then plugs into a duplicate parallel connector on the back of the box. A button on the front of the unit switches between the digitizer and the printer, making manual cable swaps unnecessary. Also on the front of the box are connectors for a microphone, a line-level audio input, and a recording volume control. An inexpensive microphone completes the package.

Connection to a sound source is simple. Most component stereo devices, like tape recorders and compact disk players, will just plug into the line-level input jack on the digitizer. You will need a Y-adapter however to record both right and left channels. The package includes an inexpensive microphone, but any microphone with a 3.5-mm plug can be used. The microphone can be connected at the same time as a line-level device, allowing mixing between the two sources. Of course, the microphone can be used by itself, too.

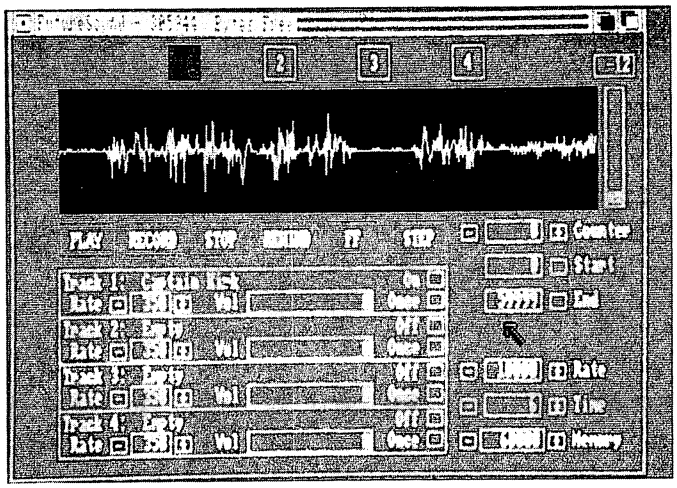
Software

FutureSound's software package is quite complete. The recording program has four tracks into which sounds may be recorded; a section of the display is dedicated to showing a graph of the sound in the current track. A peak-holding recording level meter shows signal strength and indicates when clipping is occurring; adjusting of the recording level simple.

This program is an example of an application in which the icon-style user interface excels: the main program controls look like the buttons on a tape recorder, including **RECORD, PLAY, STOP, REWIND** and **FF** (fast forward). Any, or all of the tracks can be played at one time. Playback can be one-time or repeating; sampling rate, playback speed, and volume are adjustable for each track. Sounds may have their overall amplitude (volume) scaled up or down, and can be reversed. The option to mix and copy sounds between tracks is also available. All these choices will work on an entire track, or a user-selected portion of it, making it possible to create extremely complex sounds from relatively plain samples.

The sampling program is mouse and menu-based, with control key alternatives for most selections. Numeric selections, like sampling rate and playback speed, are modified by clicking the mouse on "+" and "-" gadgets. If you hold down the mouse button, the number changes at a faster rate. It can still take a while, though to make major changes. Being able to enter numbers directly, in addition to the

continued...



gadgets, would improve speed of operation. On the whole, though, this program is extremely well-planned and executed, making references to the manual unnecessary.

Also on the disk are source code files for use in user programs, in both C and BASIC. (Incidentally, the BASIC files were written by *Amazing's* own John Foust.) These files make use of custom sounds simple, with such self-explanatory routines as LoadSound and PlaySound. In addition, the disk includes complete example programs in both languages.

Documentation

The instructions consist of a 38-page manual which is surprisingly informative and complete. The text is clear and concise, starting with a section on hardware installation and ending with a technical section which explains details and provides a phone number for technical information and assistance. I had a question that was not answered in the manual, so I tried calling this number. I was told that the person that I needed to speak to was out, and would call me back. Rather than getting the expected runaround, I got a call back, only half an hour later. After a short explanation of my problem, he offered to send me a disk containing the necessary routines to solve it! I was pleasantly pleased, to say the least.

Using The Package

After setup, recording sounds is as easy as clicking the mouse on the **RECORD** button and saying a few words into the microphone. Recording time depends on both the sampling rate and the amount of memory allotted for a track. Using the defaults provided by the program results in ten seconds of sound recorded at a 10 KHz (kilohertz, or thousands of cycles per second) sampling rate. Sound quality is somewhat dependent on the sampling rate: faster rates provide higher quality sound, but use more memory. I found that sampling from my Akai component stereo tape deck at the 10 KHz default rate resulted in good quality sound, at a reasonable rate of memory usage. Using a graphic equalizer to "pre-equalize" sounds, amplifying both bass and treble sections, should result in even better quality. In addition, the software will allow recording at rates all the way from 1 KHz to 28 KHz, the Amiga's maximum rate.

I was surprised by the quality of the recordings made with the package's microphone. Despite its rather cheap appearance, it performed quite well.

Complaints

FutureSound's only real problem is with the printer/ digitizer selector button on the digitizer box. This button has a small red light which lights up when the digitizer is on. Simple, right? But, when the computer is first turned on, the button is dark, even though the digitizer is actually enabled. Pressing the button once disables the digitizer, and selects the printer at the same time. After that, the light works correctly. Using a simple toggle switch, instead of the lighted button, would give a correct visual feedback of the status of the device. Still, it is not a difficult problem to live with, and it beats switching cables repeatedly. I simply press the button a couple of times when I turn the computer on.

It would also be nice if the recording software allowed the user to edit the sound sample by "drawing" in its graphic representation with the mouse. There are other commercial packages available for this purpose, though.

Conclusion

FutureSound solves the problem of digitizing sounds on the Amiga in style. FutureSound has a complete and readable manual and comes with everything you need to start using it right away. Applied Visions' customer support is good, too. Developers, musicians, and anyone who wants to use the Amiga's sound capability will not go wrong by purchasing FutureSound.

•AC•

FutureSound Sound Digitizer Package
List Price: $175

Applied Visions
Suite 2200
One Kendall Square
Cambridge, MA 02139

REAL Stereo Sound Effects

Using FutureSound™ with AmigaBASIC™

by Jim Meadows

Compuserve: 75046,2012
People Link: OPS321

If you want to "spice up" your basic programs with real, digitized sound effects in stereo, FutureSound may be for you. In this article I describe using FutureSound with Basic and techniques for producing stereo sound effects. I developed the accompanying sample program, STEREO, to give you an example to work with.

Set-up

To hear the stereo effects you must have your Amiga connected to a stereo amplifier with two speakers. You lose the left/right distinction in sound if the left and right phono outputs of your Amiga are combined and connected to an Amiga monitor; but you can still produce very realistic sound effects.

The FutureSound digitizer plugs into the parallel printer port (and your printer then plugs into the digitizer). The red button on the front of the sound digitizer should be lit while digitizing sounds (Press it once if it is off to turn it on -- pressing the button alternates between the printer and the digitizer).

To use the Basic routines provided with FutureSound, you must first copy the Future.Library from the FutureSound disk to the LIBS directory on your Workbench disk. One way to do this from the CLI is:

```
COPY FutureSound:BasicFiles/Future.Library to
Ram:
  COPY Ram:Future.Library to Workbench:Libs
```

Next, copy the Bmaps from the FutureSound disk onto your disk that has AmigaBasic and your basic programs on it. Assuming your basic disk is called Extras, you can do this from the CLI as follows:

```
COPY FutureSound:BasicFiles/#?.Bmap to Ram:
COPY Ram:#?.Bmap to Extras:
DELETE Ram:#?
```

Before saving a sound, you must set up a sound directory on the disk you wish to save it on. Insert your FutureSound disk and click its disk icon and then click the Future program icon. Once the program is started, you can remove the FutureSound disk and insert your disk with AmigaBasic and your programs on it. If you have your basic disk in an external drive, first select "Change Data Drive" from the disk menu and indicate the drive to use (the default is DF0:). Now, select "Make Data Disk" from the Disk Menu and click on the FORMAT requestor that appears. You only do this once for your disk that has your programs on it (don't worry, it won't wipe out your Basic disk; it just adds a directory called SoundFiles to save the digitized sound in.)

Digitizing Sounds

You are now ready to develop your digitized sounds. Select one of the 4 tracks to record in, point and click on RECORD. FutureSound begins recording from the microphone if it is plugged in, or from the phono input if something is plugged in there. The length of the recording, as well as the sampling rate, may be changed at the bottom of the screen . The recording volume can be adjusted using the knob on the front of the FutureSound digitizer.

Once a sound has been recorded, its digitized waveform is shown at the top of the screen. By pointing and clicking at PLAY you can hear how it sounds. Using the editing commands, you can select just a portion of the sound recorded and save it, combine it with other sounds, scale it, reverse it, etc. By using the magnify bar, you can zoom in on a portion of the waveform and mark the exact points for the beginning and end of the section of the recording you wish to save. The FutureSound manual describes these procedures in detail. **Note:** When you save a track or a portion of a track you just recorded, the title of the track remains "Untitled". Later, if you load the sound from disk into a track, THEN its file name appears above the track as its title.

continued...

Using Digitized sound in Basic

To use the sample program listed with this article you need to record 5 sounds called "Explosion", "Engine", "Siren", "Shoot", and "Ready". You should record sounds that resemble these names. Just say READY for the "Ready" file. The Engine file should contain a continuous sound, with no silent portions. Before recording each sound, point and click on the Time "-" indicator at the bottom right of the screen and reduce the time to 1 second. This procedure will conserve memory, since it takes 10,000 bytes for 1 second at the standard 10K sampling rate. (You get longer recording times with less memory if you use a slower sampling rate, but the quality begins to degenerate.)

If you don't have a VCR tape with the appropriate sounds on it, you can improvise with your own built-in sound effects generator. (Caution: Do this alone to avoid getting strange looks while you sit in front of your computer holding a microphone saying "Bpkrrrrghhhhhh").

After recording and saving the 5 files onto your Basic disk, close the FutureSound windows and start up AmigaBasic. Now, how do you get these sounds off disk into memory and use them from Basic? John Foust developed some nice routines that are on the FutureSound Disk to accomplish this task. I condensed and modified them to produce the stereo sound effects routines shown at the end of the sample program. You will probably want to look at John Foust's listing and read his comments to get a better understanding of the FutureSound functions.

Sample STEREO Program

Now let's look at the STEREO sample program listing. The program begins by setting NumSnds to the number of digitized sounds that are going to be used (5, in this case). Next, the appropriate functions, libraries, and arrays are initialized by the routine SetUp (you only want to do this once in a program). To load a sound from disk into memory, you set SoundName$ to the file name, SndNum to the sound reference number, and call LoadSound. The routine InitSounds does this for all 5 sounds. When a sound is loaded, its memory pointer, length, and pitch is saved to later activate the sound. In the sample program, these facts are saved in the arrays SndMem&(N), SndLength&(N), and SndPitch&(N) where N is the reference number for each sound. (1 for Explosion, 2 for Engine, etc.). **Note**: Sounds must be loaded into the lower 512K chip memory for them to work properly. For the basic routines supplied with FutureSound to work properly on systems having more than 512K, you must change the value MemType& is set to. Insead of MemType&=65537& use MemType&=65538&.

Now for the fun part. To play one of the digitized sounds, you set SndNum to the reference number of the sound you wish to play (e.g. Set it to 3 to play the sound Shoot). Set ChanNum to the channel you want to use (LeftA, RightA, LeftB, or RightB where A denotes primary channels 0 & 1 and B indicates secondary channels 2 & 3). Then, set Vol& to the volume desired (0-64) and call PlaySound to make a real digitized sound emerge from your Amiga! The sound will play once and stop, unless you change Reps& to a value other than 1. A value of 0 for Reps& will cause the sound to repeat until another sound is played for that channel. A value greater than 1 repeats the sound that many times. Just set these parameters and call PlaySound -- that's all there is to it! As you look through the listing, you will see that the program consists primarily of determining the values of the parameters needed to produce the various stereo effects.

Now run the STEREO sample program. The program first plays sound 5 (Ready) out of both speakers. After a pause, sound 3 (Siren) is repeated 10 times, varying in volume from left to right (it sounds like it passes by you!). Next, all 4 channels are used together. The Engine sound is started up in both the left and right primary channels. Pressing the up arrow key increases the sound pitch, while pressing the down arrow decreases it. The left arrow key increases the left channel sound volume while decreasing the right. The right arrow key will have the opposite effect. Pressing the space bar will cause the Shoot sound to come from the secondary left channel and, after a moment, the Explosion will sound from the secondary right channel. The Engine sound will continue playing uninterrupted, since it is playing on the primary channels. If you press the wrong key, you will be "told" of your error through the AmigaBasic Say command. You exit from this mode by pressing the Esc key. The program then says "Good bye", frees up the memory that was reserved for the digitized sounds, closes the libraries and ends.

How it works

The PlaySound routine takes the sound number passed to it and sets the memory pointer, length, pitch, and identifier for the selected sound. Then, PlaySound calls the FutureSound routine FSStopSound&, followed by FSPlaySound&, using the Reps&, Pitch&, and Vol& parameters. You may wonder why you need to use FSStopSound& before FSPlaySound&. FSPlaySound& plays a digitized sound in the next available channel. For stereo sound effects, you need to control which channel the sound will come from. The C routines could be modified to accomplish this control, but since I did not have a C compiler for the Amiga, I developed my own technique using the existing routines.

If you look back at the routine InitSounds, you will find that after loading the 5 sounds, I started up a sound in each of the four channels, using a volume of 0. Knowing the sequence the sounds are initially assigned in (Left, Left, Right, Right) allows the sound identifier, returned by FSPlaySound&, to then be associated with a specific channel. The PlaySound routine first stops the sound associated with the channel you wish to use. This is done so that when you call FSPlaySound&, it will use that channel, since it is the only one available.

This same technique is used by the SayPhrase to use the AmigaBasic Say command. Since all 4 channels are always occupied by a digitized sound, the Say command would not have any channel to use. Therefore, SayPhrase first stops the sound associated with the channel you wish to use. SayPhrase then uses the Say command with a channel parameter set to use any available single Channel (How%(6)=11) to produce the phrase passed to the routine. After the phrase has been spoken, a digitized sound with 0 volume is reassigned back to that channel.

Other techniques

Another technique I used in the game Gemini-2, not shown in the sample program, is combines all the digitized sounds into a single file using the FutureSound edit functions. As I built the file, I noted the offset of each sound from the beginning of the buffer, as well as its length. I put these values in the program as a data array and use them to compute the memory pointer and length, based on the sound number. This process makes changing the sounds more difficult, but allows a single load to load all the sounds at once. It also allowes you to combine adjacent sounds in the buffer into one long sound, by varying the length parameter. Playing sounds back at pitches considerably different from how they were originally recorded also allows you to get multiple uses out of the same sound.

Conclusion

I hope the STEREO sample program accomplishes "one example is worth a thousand words." If you didn't understand some of the whys in this article, don't worry about it. Just set the parameters discussed and call the routines as shown in the sample program and you can add great stereo sound effects to your Basic programs using digitized sounds!

continued...

```
' STEREO - A sample program for using
' FutureSound Digitized sound effects
' in Stereo with AmigaBasic
'
'    by Jim Meadows -- 2/19/87

' First Specify the number of sounds
NumSnds=5

' Then Setup Library Functions
PRINT:PRINT "Setting up Functions ...."
GOSUB SetUp

' Now Initialize the Sounds
PRINT:PRINT "Loading Sounds ...."
GOSUB InitSounds
PRINT:PRINT "Press left mouse button"
Pause:
  IF MOUSE(0)=0 THEN GOTO Pause

' Play READY in both speakers
PRINT:PRINT "Ready":PRINT
SndNum=5:Vol&=64
ChanNum=LeftA:GOSUB PlaySound
ChanNum=RightA:GOSUB PlaySound
FOR I = 1 TO 8000:NEXT

' Play SIREN from left to right
CLS
SndNum=3:L=64:Reps&=0:R=4
FOR S = 1 TO 10
  LOCATE 10,1:PRINT SPACE$(S*7)+"siren"
  Vol&=L:ChanNum=LeftA
  GOSUB PlaySound
  Vol&=R:ChanNum=RightA
  GOSUB PlaySound
  L=L-6:R=R+6
  FOR I=1 TO 3000:NEXT
NEXT
FOR I = 1 TO 3000:NEXT
CLS

' Now start ENGINE in primary channels
SndNum=2:Vol&=32:Reps&=0
ChanNum=LeftA:GOSUB PlaySound
ChanNum=RightA:GOSUB PlaySound
L=32:R=32

' Vary Pitch and Volume
' and Shoot using keypresses
' Say Wrong Key if necessary
CLS:LOCATE 10,1
PRINT "Press Arrow Keys or Space Bar"
PRINT "for Stereo Sound Effects"
PRINT "Press Esc to Quit"
CkKey:
 K$=INKEY$:IF K$="" THEN GOTO CkKey
 IF K$=CHR$(27) THEN GOTO Esc
 IF K$=CHR$(28) THEN GOTO Up
 IF K$=CHR$(30) THEN GOTO Right
 IF K$=CHR$(29) THEN GOTO Down
 IF K$=CHR$(31) THEN GOTO Left
 IF K$=" " THEN GOTO Shoot
 Phrase$=TRANSLATE$("Wrong, Key")
 ChanNum=RightB:GOSUB SayPhrase
GOTO CkKey

 Up:
   SndPitch&(2)=SndPitch&(2)-20
   GOTO ChangeEngine

 Down:
   SndPitch&(2)=SndPitch&(2)+20
   GOTO ChangeEngine

 Left:
   L=L+5:IF L>64 THEN L=64
   R=R-5:IF R<0 THEN R=0
   GOTO ChangeEngine

 Right:
   R=R+5:IF R>64 THEN R=64
   L=L-5:IF L<0 THEN L=0
   GOTO ChangeEngine

 ChangeEngine:
   SndNum=2:Reps&=0
   Vol&=L:ChanNum=LeftA
   GOSUB PlaySound
   Vol&=R:ChanNum=RightA
   GOSUB PlaySound
   GOTO CkKey

 Shoot:
   Vol&=64:Reps&=1
   SndNum=4:ChanNum=LeftB
   GOSUB PlaySound
   FOR I=1 TO 6000:NEXT
   SndNum=1:ChanNum=RightB
   GOSUB PlaySound
   FOR I=1 TO 3000:NEXT
   GOTO CkKey

Esc:
  PRINT:PRINT "Good Bye"
  Phrase$=TRANSLATE$("Good Bye")
  ChanNum=LeftB:GOSUB SayPhrase
  GOSUB StopSounds
  GOSUB CleanUp
 END:' End of Program



InitSounds:
 SndNum=1:SoundName$="Explosion"
 GOSUB Loadsound
 PRINT "Explosion Loaded"
 SndNum=2:SoundName$="Engine"
 GOSUB Loadsound
 PRINT "Engine Loaded"
 SndNum=3:SoundName$="Siren"
 GOSUB Loadsound
 PRINT "Siren Loaded"
 SndNum=4:SoundName$="Shoot"
 GOSUB Loadsound
 PRINT "Shoot Loaded"
 SndNum=5:SoundName$="Ready"
 GOSUB Loadsound
 PRINT "Ready Loaded"
 GOSUB AssignChan
RETURN

' ------------------------------------
' Stereo Sound Effects Routines
' ------------------------------------
' The following routines can be added to
' a Basic program to play digitized
' sound effects in Stereo.  These routines
' use FutureSound Functions.

SetUp:
' This routine should be called 1 time
' Initialize library functions
  DECLARE FUNCTION FSGetSize& LIBRARY
  DECLARE FUNCTION FSLoadSound& LIBRARY
  DECLARE FUNCTION FSPlaySound& LIBRARY
  DECLARE FUNCTION FSStopSound& LIBRARY
  DECLARE FUNCTION AllocMem& LIBRARY
  LIBRARY "future.library"
  LIBRARY "exec.library"
' Define arrays for sound parameters
' (Set size to number of sounds used)
```

continued...

ATTENTION! DELUXE MUSIC CONSTRUCTION SET™ USERS

ATTENTION! MUSIC STUDIO™ USERS

SYMPHONY SONGS

ATTENTION! INSTANT MUSIC™ USERS

SYMPHONY SONGS gives you a library of nearly 1,000 music masterpieces. All songs are in IFF format so they may be loaded, played, printed, transposed, and modified in any way you like using your favorite composition program. Included is a free program to convert the IFF files to MUSIC STUDIO™ format.

The songs have been arranged by C. Clark Rulaford and Randy Spector and take advantage of the full 4 Voice capability of the AMIGA.

Space does not allow listing all the songs in each of the volumes. We have listed a few and show the total number in each volume as well as the playing time. A complete list of songs may be purchased for **$3.95.** Each volume of the 27 volumes listed is **$24.95.**

BEATLES Part 1
Vol 15 (21 Pieces 40 Min)
Let It Be, Yesterday, Eleanor Rigby, When I'm 64, . . .

BEATLES Part 2
Vol 40 (17 Pieces 40 Min)
Magical Mystery Tour, Lucy In The Sky With Diamonds, Penny Lane, . . .

CLASSICAL Part 1
Vol 27 (19 Pieces 40 Min)
Prelude #1, Moonlight Sonata 1st and 2nd Movement, . . .

CLASSICAL Part 2
Vol 34 (15 Pieces 40 Min)
Sonata In C Major, Jesus Joy Of Man's Desire, . . .

CLASSICAL Part 3
Vol 31 (18 Pieces 35 Min)
1st Piano Concerto, Polonaise, Sonata In C Major, Etude #3, . . .

CLASSICAL Part 4 (Bach)
Vol 35 (22 pieces 30 Min)
Two Part Invention #1, Three Part Invention #6, Prelude and Fugue 1, . . .

CLASSICAL Part 5 (Bach/Clementi)
Vol 46 (24 Pieces 50 Min)
Choral #1, Sonata #1, Theme and 10 Variations From The 2nd Sonata, . . .

BEETHOVEN, BROADWAY, & BLUES
Vol 38 (15 Pieces 40 Min)
2nd Movement Of the Pathetique Sonata, Minuet In G, Fuer Elise, . . .

COUNTRY CLASSICS Part 1
Vol 41 (18 Pieces 45 Min)
Thank God I'm a Country Boy, Act Naturally, . . .

ROCK Part 1
Vol 32 (19 Pieces 50 Min)
AXEL F, Eye Of The Tiger, Both Sides Now, . . .

ROCK Part 2
Vol 16 (21 Pieces 40 Min)
Georgy Girl, Guantanamera, Theme From "Love Story," Cherish, . . .

80's GREATEST
Vol 24 (16 Pieces 50 Min)
Hill Street Blues Theme, Chariots Of Fire Theme, Dynasty Theme, . . .

70's GREATEST
Vol 12 (21 Pieces 45 Min)
Tie A Yellow Ribbon On The Old Oak Tree, We've Only Just Begun, . . .

60's GREATEST
Vol 13 (21 Pieces 45 Min)
Windy, By The Time I Get To Phoenix, Come Saturday Morning, . . .

GOLD & PLATINUM HITS
Vol 45 (19 Pieces 60 Min)
Thriller, 99 Luft Balloons, California Girls, . . .

KENNY RODGERS HITS
Vol 39 (12 Pieces 45 Min)
Lady, Ruby, She Believes In Me, The Gambler, . . .

BILLY JOEL GREATEST HITS
Vol 43 (17 Pieces 65 Min)
Piano Man, Say Goodbye To Hollywood, Only The Good Die Young, . . .

COUNTRY CLASSICS Part 2
Vol 42 (19 Pieces 50 Min)
Ode To Billy Joe, Me and Bobby McGee, Country Roads, . . .

TV THEMES
Vol 37 (21 Pieces 35 Min)
Hill Street Blues, St. Elsewhere Theme, Masterpiece Theater Theme, . . .

MOVIE THEMES
Vol 19 (23 Pieces 40 Min)
M*A*S*H Theme, The Rose, Can You Read My Mind (Superman), . . .

BROADWAY'S THEMES
Vol 47 (25 Pieces 65 Min)
The Last Supper, Dr. Doolittle, The Old Dope Peddler, . . .

CHURCH MUSIC
Vol 28 (26 Piece 50 Min)
Amazing Grace, What A Friend We Have In Jesus, . . .

BARBERSHOP
Vol 22 (22 Pieces 45 Min)
Hello Dolly, Put On a Happy Face, Hey Look Me Over, . . .

RICHARD RODGERS SONGBOOK
Vol 18 (19 Pieces 40 Min)
Climb Every Mountain, DO-RE-MI, The Sound Of Music, . . .

NOSTALGIA
Vol 17 (22 Pieces 45 Min)
Let Me Call You Sweetheart, Ain't Misbehavin', On The Goodship Lollipop, . . .

CHRISTMAS
Vol 36 (26 pieces 50 Min)
O Little Town Of Bethlehem, Let It Snow, March Of The Toys, . . .

POLKA PARTY
Vol 33 (18 Pieces 40 Min)
Happy Polka, Pizzacato Polka, Betty Polka, . . .

SYMPHONY JUKEBOX

Symphony Jukebox allows you to program a selection of songs, their order as well as the number of times, and allows you to listen to them for hours of uninterrupted playing. Other features include MIDI output, instrument selection, transposition, and tempo modification. **$24.95**

SYMPHONY MUSIC VIDEO

This program has all the features of our SYMPHONY JUKEBOX, however, it also allows you to specify a picture to be displayed with each song. The pictures and music are all in standard IFF format so you may use the songs and pictures included, or use those you developed with your music program (i.e. DMCS), or your paint program (i.e. Deluxe Paint). Included are Christmas music and pictures. **$24.95**

We accept CASH, CHECK, COD, VISA and MASTER CARD orders.
Shipping and handling US and Canada $3.00
Shipping and handling outside the US and Canada $5.00
COD charge $2.00
Illinois residents add 6¼% sales tax.

VISA MasterCard
Dealer Inquires Invited

Speech Systems
38W255 DEERPATH ROAD
BATAVIA, ILLINOIS 60510
(312) 879-6811

```
  DIM SndMem&(NumSnds)
  DIM SndLength&(NumSnds)
  DIM SndPitch&(NumSnds)
  DIM SndID&(NumSnds)
' Say parameters (See SAY in AmigaBasic
' Reference Manual section 8)
  DIM How%(8)
  FOR I = 0 TO 8:READ How%(I):NEXT
  DATA 110,0,150,0,22000,64,11,0,0
'                            --
RETURN

Loadsound:
' Load a sound into memory
 SoundName$ = "SoundFiles/"+SoundName$+CHR$(0)
 Length& = FSGetSize&(SADD(SoundName$))
 IF Length&=0 THEN
   PRINT "Not Found":GOTO NoSound
 END IF
 MemType& = 65538&:' (Chip Memory)
 Mem& = AllocMem&(Length&,MemType&)
 IF Mem& = 0 THEN
   PRINT "Memory Full":GOTO NoSound
 END IF
 SndMem&(SndNum)=Mem&
 SndLength&(SndNum)=Length&
 Rate& = FSLoadSound&(SADD(SoundName$),Mem&)
 IF Rate& = 0 THEN
  PRINT "Invalid Load":GOTO NoSound
 END IF
 SndPitch&(SndNum) =  INT(3579545& / Rate&)
' Note: Decreasing SndPitch& raises the
' pitch of the sound produced
RETURN

NoSound:
 PRINT  SoundName$" not loaded."
 GOSUB CleanUp
 END:' End Program

AssignChan:
' Assign channel ID's by starting
' a sound with 0 volume in each channel
 LeftA=0:RightA=1:LeftB=3:RightB=2
 Reps&=1:Vol&=0
 Mem&=SndMem&(1)
 Length&=SndLength&(1)
 Pitch&=SndPitch&(1)
 FOR I = 1 TO 4
  ID&=FSPlaySound&(Mem&,Length&,Reps&,Pitch&,Vol&)
  IF I=1 THEN SndID&(LeftA)=ID&
  IF I=2 THEN SndID&(LeftB)=ID&
  IF I=3 THEN SndID&(RightA)=ID&
  IF I=4 THEN SndID&(RightB)=ID&
 NEXT
RETURN

PlaySound:
' Play Sound indicated by SndNum
' in channel indicated by ChanNum
  Mem&=SndMem&(SndNum)
  Length&=SndLength&(SndNum)
  Pitch&=SndPitch&(SndNum)
  ID&=SndID&(ChanNum)
  Halt&=FSStopSound&(ID&)
  ID&=FSPlaySound&(Mem&,Length&,Reps&,Pitch&,Vol&)
  SndID&(ChanNum)=ID&
RETURN

SayPhrase:
' Say Phrase$ in channel indicated
' by ChanNum
  ID&=SndID&(ChanNum)
  Halt&=FSStopSound&(ID&)
  SAY Phrase$,How%
  Vol&=0
  ID&=FSPlaySound&(Mem&,Length&,Reps&,Pitch&,Vol&)
  SndID&(ChanNum)=ID&
RETURN

StopSounds:
' Stop all sounds
 FOR I = 0 TO 3
  ID&=SndID&(I)
  Halt&=FSStopSound&(ID&)
 NEXT
RETURN

CleanUp:
' Free up sound memory and Close library
 FOR I=1 TO NumSnds
  Mem&=SndMem&(I)
  Length&=SndLength&(I)
  IF Mem&<>0 THEN
    CALL FreeMem(Mem&,Length&)
  END IF
 NEXT
 LIBRARY CLOSE
RETURN
```

•AC•

Programming with MIDI, the Amiga, and SoundScape

Writing a SoundScape Module in C

By Todor Fay
Author of SoundScape

With MIDI and SoundScape, you can write programs that manipulate musical information. In this article, I'll explain what MIDI and SoundScape are, then give two example programs that manipulate music from within the SoundScape environment.

An Introduction to MIDI

The last five years have seen a revolution in the music studio. This revolution is MIDI.

In today's music studio, many devices are connected with MIDI (Musical Instrument Digital Interface) cables. Keyboards, rack mount synthesizers, drum machines, sequencers, effects boxes, even automated mixers, are strung together by MIDI.

MIDI provides a method of communication between these devices. It enables a keyboard to tell a rack mount synthesizer when a key has been hit so it can play it. MIDI enables a drum machine to synchronize with a sequencer by having the drum machine tell the sequencer when to start, and how fast to play.

MIDI is organized as a one way message service. A source sends MIDI events to one or more receivers. The receivers do not reply; they simply read the events and process them if appropriate.

MIDI events are usually three or fewer bytes in length. The first byte is called the Status byte. The Status defines the type of event. It is followed by the number of data bytes necessary to convey this data (sometimes none.) The data bytes always have their MSB cleared. This clearing gives only seven bits for the data, but it identifies that this is a data byte, not status. Status always has the MSB set.

One example would be a note on event. There are three bytes in such an event. The first byte, the status, has the Note On code. The second holds the value of the note, a number between 0 and 127. The third holds the velocity of the note, a number between 1 and 127. The higher the velocity number, the faster the key was struck. This usually translates into a higher volume.

Some MIDI event types have a destination id built into the event. This id allows several different devices to be received from the same source, but each can be designated to listen to different events by assigning different ids to the receivers. The id is a four bit number in the lower four bits of the status byte. This gives us sixteen ids, from 0 to 15. Each MIDI receiver can be set to listen only to events which are intended for one particular id. So, with sixteen ids available, one source can send to sixteen receivers and each of them will play different notes. Of course, it is also perfectly legal to have two receivers listen for the same id; they'll just both play the same notes.

In MIDI terminology, these ids are called 'channels'.

Events that carry channel information are all the events that are used to transmit a musical performance. For each of these events, the upper four bits of the status byte actually define the command, while the lower four bits define which channel the event is intended for.

Here's a run down of these events:

NOTEON:

NOTEON is sent whenever a note is to be played. There are three bytes. As with all events, the first byte carries the status. Since this is a channelized event, the channel id is stored in the lower four bits of the status, while the upper four bits carry the NOTEON status. The second byte carries the note value. The third byte holds the velocity. If the velocity is 0, this note is actually a note off event, the note should stop being played.

NOTEOFF:

NOTEOFF is the same as NOTEON, except it is a command to stop playing the specified note. This command also has a velocity byte which notes how fast the finger lifted up when the key was released. Many manufacturers use the NOTEON event and a velocity of zero for note off, so be prepared to handle it.

continued...

POLYPRESSURE:
Some keyboards are capable of measuring how hard each individual key is being pressed. POLYPRESSURE is used to transmit that information. There are three bytes, including status. The second holds the value of the note, just as with NOTEON. The third byte holds the pressure, a number between 0 and 127.

AFTERTOUCH:
AFTERTOUCH is the same as POLYPRESSURE, except it is mono; it refers to all the keys at once. So, there are only two bytes - the status and the overall pressure.

PITCHWHEEL:
Every time the pitchbend wheel on a synthesizer is moved, this event is sent. Including the status, there are three bytes. The absolute position of the wheel is broken into two seven-bit segments. The least significent bits are in the second byte, the most significant in the third.

CONTROLCHANGE:
In addition to pitch bend and after touch, there are lots of other performance parameters which can be sent. One example is a modulation wheel. This is a catch-all event that can be used for sending all types of data. There are three bytes. The second is the control change number - a value between 0 and 123. This event specifies the type of event. The third byte is the actual data.

PROGRAMCHANGE:
Most synthesizers have a bank of preset sounds which the musician can instantly switch between. This event has two bytes - the status and the number of a preset.

The remaining MIDI events are not channelized. All devices respond to these events, if they can. For example, there are several timing commands which control devices such as sequencers and drum machines which need timing information in order to run. We'll talk about those later on.

Although MIDI is intended primarily as a method for transmitting performance information between a music source and performers, it also provides a great opportunity to process musical information in real time. In other words, there's nothing to stop you from intercepting the MIDI events with a black box that does something weird and wonderful with them, then passes them on.

For example, there are products you can buy in music stores that will translate one type of event into another. PITCHWHEEL becomes fast note arpeggios. There are echo devices that work by simply storing an event and sending multiple delayed copies of it.

There are products that perform along with MIDI in unanticipated ways. Lighting controllers that switch lights on PROGRAMCHANGE events. Automated mixing consoles that can have volume levels, effects, and even equalization controlled by CONTROLCHANGE events. Devices that can be controlled by MIDI have a wonderful advantage. Not only can they be remotely controlled, but this can be done as part of a musical performance.

Just think what could be done with an Amiga!

SoundScape and MIDI
SoundScape provides an environment that is similar to a MIDI studio. Taking advantage of the Amiga's multitasking ability, SoundScape lets any number of programs behave as if they were MIDI devices, each capable of sending and receiving MIDI events. Each of these MIDI devices, or modules, is represented by an icon in the primary SoundScape window - the Patch Panel.

In the Patch Panel, modules which create MIDI events are displayed on the left. Modules that receive MIDI events are displayed on the right. The user makes MIDI "patches" by clicking on a left icon, then a right icon. A line is drawn between the two modules, indicating that the patch has been made. One module can send to multiple receivers, and multiple modules can send to one receiver.

Once these connections have been made, any MIDI event generated by a module on the left will be sent to all modules that receive from it on the right.

To handle moving these MIDI events from left to right, SoundScape provides its own message-passing service. Unlike the Amiga's message passing, this is a datagram service, i.e. there is no replying, no handshaking. This is so because MIDI is a datagram service, and with good reason: the nature of real time music information places much more emphasis on throughput and response time, than on accuracy and synchronization. This service also makes it very easy for one source to send to multiple receivers.

Each module that is represented in the patch panel has a 'port' for sending and receiving MIDI events. Some ports can only send, some can only receive, and some can do both. Each port has a unique identifier, a number between 1 and 255.

It is possible for an outside program to install itself as a port in the Patch Panel and become part of the SoundScape environment.

In other words, you can write your own MIDI software and integrate it with the SoundScape system.

How is this done?
SoundScape exists on the Amiga as a library. So, SoundScape is invoked by one or more programs calling OpenLibrary ("soundscape.library",0). As a library, SoundScape allows multiple programs to interface with it through library function calls. These calls include routines that a module uses to install itself in the Patch Panel, and routines to create, send and receive MIDI events.

Here's a run down on the tools you will use to write your own MIDI device:

Data Structures

Just about everything in SoundScape is done with linked lists. MIDI note packets are queued by the router in linked lists. The Tape Deck keeps a linked list of tracks, each pointing to a linked list of notes - the sequence. The memory allocator keeps linked lists of freed list nodes, so that nodes can be allocated very quickly without resorting off the heap.

Each list node type starts with the same generic header:

```
struct Link {
    struct Link *next;  /* Next node in the linked list. */
    unsigned char type;             /* Type of this node. */
    unsigned char mark;
    unsigned short data;
};
```

All of the fields in the Link structure should be left alone.

Here is the Note structure:

```
struct Note {
    struct Link link;
    unsigned short duration;/*Clock beats the note is on.*/
    unsigned short wait;  /* Clock beats till next note. */
    unsigned char status;       /* Midi status. */
    unsigned char value;        /* First byte. */
    unsigned char velocity;     /* Second byte. */
};
```

Note nodes are actually the same as MIDI packet nodes. They carry MIDI status, note value and note velocity (bytes one and two of the standard MIDI packet.) For sequencing, two fields, duration and wait, are also included. These features specify how long the note is played and how long to wait before playing the next one.

Allocating and Sending MIDI Events

Because the SoundScape MIDI event routing system is a one way datagram service, there needs to be some method for allocating and returning these MIDI events. With the Amiga's Exec message passing system, this process is handled by having the receiver of the message always reply. This way, the message is returned to the message originator which can reuse it or deallocate it.

But with MIDI events in SoundScape, the event is sent to possibly multiple (or no) receivers, and each may deal with it differently. Some may actually store the event for later use (which is what a sequencer, like the Tape Deck, would do.). Others may alter the event and send it on to yet other modules.

Two routines, AllocNode() and FreeNode(), let you allocate MIDI events from the central allocator and return them when finished.

When a module is ready to send a MIDI event, it calls AllocNode(NOTE) to get that event. AllocNode can actually allocate one of three different kinds of structures: NOTE, SEQUENCE, and TRACK. The SEQUENCE and TRACK structures are useful only to the Tape Deck. The NOTE structure is also designed for use in the Tape Deck, which is why two of its fields, 'duration' and 'wait' are not part of a normal MIDI event. But, since it is these MIDI events that end up being tied together into a sequence by the Tape Deck, it is easiest to use the same structure for both.

Here's the code to allocate a MIDI event:

```
struct Note *midievent;
  midievent = (struct Note *) AllocNode(NOTE);
```

If AllocNode returns NULL, there's not enough memory left in the system. You should always be prepared for this problem.

Stuff the status code and appropriate data into the Note structure, then send it with the Send command:

```
if (midievent) {
    midievent->status = NOTEON;
    midievent->value = 90;
    midievent->velocity = 64;
    Send(thisport,midievent);
}
```

continued...

For this example, we sent a note on event. The first data byte, the note value, was 90 (octave 7, F#). The second data byte, the velocity, was 64. 'Thisport' is the id of the port for this module. The Send command instructs SoundScape to take 'midievent' and send copies of it to all ports that are set up to receive from 'thisport'.

The Send command places 'midievent' in a queue of MIDI events. This queue is processed by a task that runs at a slightly high priority. This task, the MIDI event router, takes each event, makes copies of it, and passes each copy to a port that is supposed to receive the event. Send passes the copy to each port by calling a routine which the port provides, and passing the event as a parameter.

Receiving MIDI Events

So, each port must provide a routine for handling MIDI events sent to it. Here's an example of such a routine:

```
outcode(midievent)

struct Note *midievent;

{
    FreeNode(midievent);
}
```

This sample does absolutely nothing with the event. It just returns it to the central allocator.

Open, Close, and Edit routines

In addition to the output routine, there are three other routines which each port provides. There is a routine for opening the port, a routine for closing it, and a routine that lets the user or other software edit its parameters.

The open routine is called whenever someone wants to activate the port. For example, the first time the user connects this port to another port on the patch panel, this routine is called. Often, this routine does nothing; it depends on the module. For example, when the Console Keyboard module is opened, the open routine creates a task. This task puts up a window that displays a keyboard and then waits for keys to be hit. When keys are hit, this task will allocate MIDI events and send them.

On the other hand, the MIDI mixer does nothing in its open routine.

The only thing an open routine must do is return 1 if successful, 0 if unsuccessful. So, if the console keyboard can't create the task which opens the window, it returns 0.

A port can be opened for input and output seperately. So, the open routine is passed one parameter, a flag that is set for opening for MIDI event sending (icon on left side of Patch Panel) and cleared for opening for MIDI event receiving (the right side.)

Here's an example open routine for a port which only wants to receive MIDI events:

```
opencode(direction)

unsigned char direction;

{
    if (direction) return(0);
    else return(1);
}
```

The close routine is very similar to the open routine, only it is used to deactivate the port for one particular direction. For example, the Console Keyboard will shut down the window and delete the task when its close routine is called.

A close routine that works with the above open routine example would be:

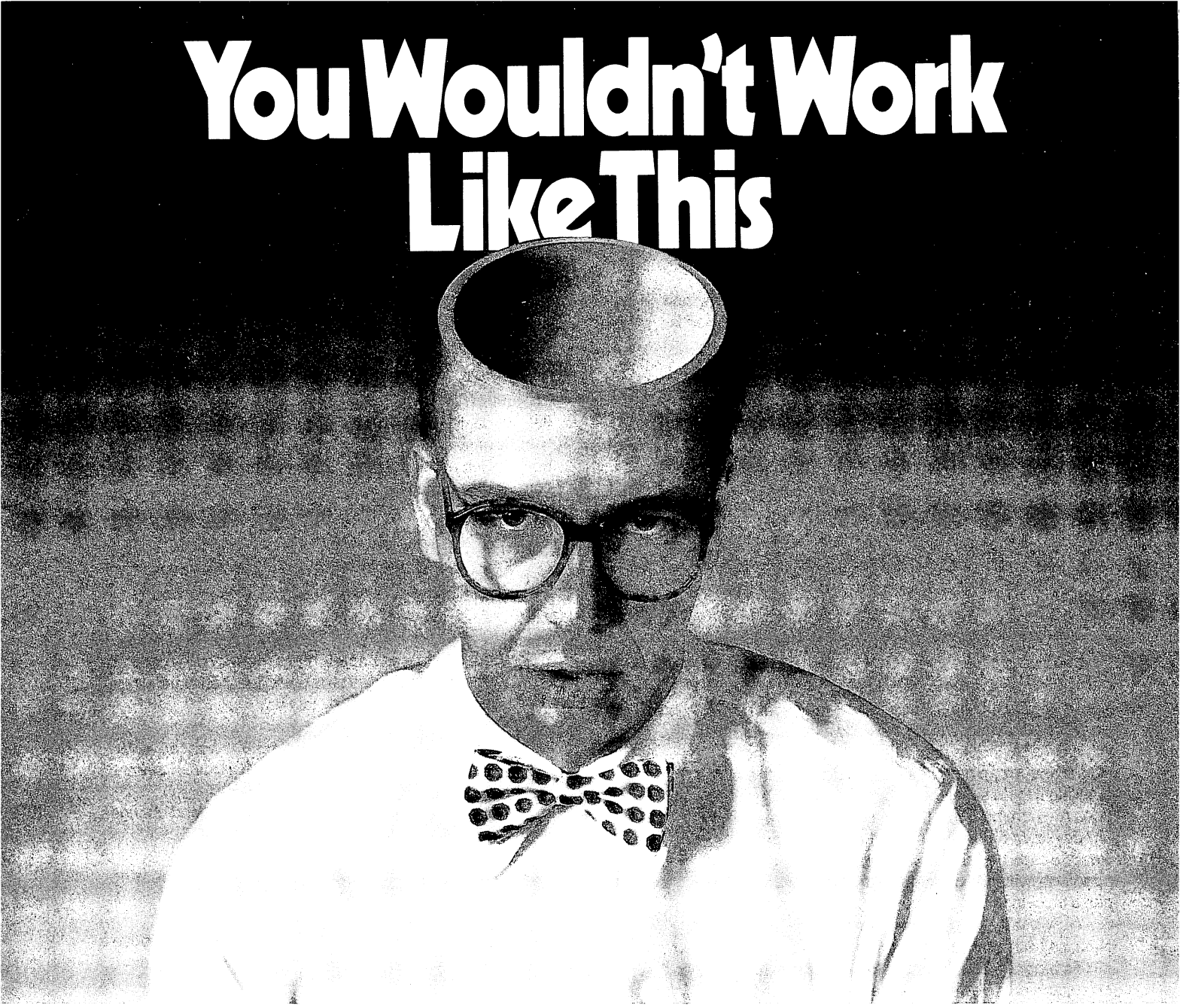
```
closecode(direction)

unsigned char direction;

{
    return(1);
}
```

Like the open routine, the close routine should return 1 if successful, 0 if unsuccessful. The close routine will only be called if the port has been opened in that particular direction. So, the above example doesn't bother worrying about the direction and always returns 1.

For now we will ignore the edit routine.

continued...

You Wouldn't Work Like This



Why Should Your Amiga?

These days an Amiga with only 512K memory is operating at a fraction of its' potential.

Fortunately we're here to change all that.

We're Expansion Technologies, and we've developed the best RAM expansion board available for your Amiga 1000.

The Escort 2.

It's a 2 megabyte, auto-configuring card that meets all the known standards and then some.

It's also a uniquely designed two-slot card cage that offers incredible flexibility.

Like the ability to upgrade to a whopping 4 megabytes of memory. Or if you prefer you can add a hard disk controller card, or an 86-pin buss return or...well, you get the idea.

And it's fast.

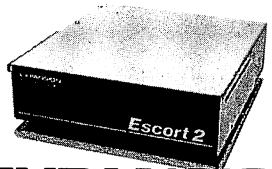
We've utilized a no wait-state design so it keeps perfect pace with your Amiga.

All this for less than $650.

We're even readying products for the new 500 and 2000 series Amigas. Among them hard drives, controller cards and memory expansion.

All of this from Expansion Technologies. A company whose people have been making Commodore products for years, not weeks.

So if your Amiga's a bit light headed give us a call at 415/656-2890. Or write us at 46127 Landing Parkway, Fremont, CA 94538.



EXPANSION TECHNOLOGIES

Dealer Inquiries Invited.

Installing Your Module

To install the port, call:

```
AddMidiPort (opencode, closecode, editcode, outcode,
                inimage, outimage, port, name).
```

Opencode, closecode, editcode, and outcode are the four routines. You can supply 0 for editcode and outcode, if you'd like. You also pass pointers to two Intuition style Image structures. These are the icons for the sending and receiving port icons. They can be the same icon, which is usually the case. If you intend only to send or only to receive, you provide only one image and 0 for the other. You need also to give AddMidiPort the name of your module, and the port id that you wish. If the port can't use that id, it will use an available one. The id it allocates (hopefully the one you asked for) is returned.

The Module Program

Having defined the images and special routines for the module, we still need to discuss the main program. This opens SoundScape and calls AddMidiPort to install the module.

The main program is very simple. Open the SoundScape library. Like opening Intuition or Graphics, this gives you the base of the SoundScape library calls. Immediately follow by closing SoundScape. This may seem strange, but it is neccessary since this is actually a module in SoundScape. If it keeps SoundScape open, then SoundScape can never be closed, because there's no way of telling it to do so later on. Closing the library at this point does no harm because the SoundScape library should already have been opened by the program called SoundScape (which does absolutely nothing, except opening a window, opening SoundScape, waiting for you to close the window, and closing SoundScape).

Next, call AddMidiPort to install this module as a port. Now all this program can do is wait to be shut down. The open, close, and output routines are all called by other tasks. We can't exit though because AmigaDOS will pull that code from memory. So, wait for SoundScape to shut down and then remove this port. Set the task priority low and wait for this port to disappear. MidiPort(thisport) is a function that returns a number if the port exists, 0 if it doesn't exist.

```
main() {
    SoundScapeBase = OpenLibrary("soundscape.library",0);
    if (SoundScapeBase) {
CloseLibrary(SoundScapeBase);
thisport = AddMidiPort(opencode,closecode,0,outcode,i
                nimage,outimage,moduleid,"modulename");
SetTaskPri(FindTask(0),-20);
while (MidiPort(thisport)) Delay(100);
    }
}
```

Let's write an example module. This module takes notes and turns them into chords. Whenever a note MIDI event comes in, the module sends the same event out, with copies of the event transposed up a third and a fifth.

(see listing for chord.c)

Here's the with file to link this example with:

```
FROM *
Lib:Astartup.obj,*
sslink.obj,*
chord.o,*
morelib.obj
TO *
ChordMaker
LIBRARY *
Lib:amiga.lib,*
Lib:lc.lib
```

To run this file, first get SoundScape running. Then, since you probably haven't made an icon for this program, type "run ChordMaker" from the CLI. A new icon will appear in both the left and right hand columns. To test the icon, connect the Console keyboard to the right hand icon and the Player Piano or Sampler to the left hand icon. Play notes. Notice how they get converted into chords.

Now that you have the tools to receive, manipulate, and send MIDI events, take some time off and experiment. Here are just a few ideas:

- A module that inverts notes. Notes coming in are subtracted from128 and sent back out. Left-handed people might be amused by this one.
- A display program that puts different IFF pictures at the front screen, triggered by PROGRAMCHANGE events.
- A module that takes PITCHWHEEL events and changes them into notes.
- A module that randomizes the velocity of notes passing through for interesting dynamics.
- A keyboard splitter. This splitter sends all notes, above a cutoff point, on one channel, while sending the notes below on another.
- **The possibilities are endless.**

After playing around a bit, get back to this article and we'll continue.

Now that you're familiar with the simplest level of SoundScape programming, we'll dig into it a little deeper. We'll discuss MIDI real time events and write an application - a MIDI echo device - that uses these events. We'll also get into edit routines and state structures: the mechanisms for allowing the user and other programs to access your module and change parameters.

MIDI Timing Events

Along with communicating note and performance information, some MIDI devices need to synchronize in time. For example, one piece of music could be performed by a

sequencer and drum machine, in tandem. The two machines must start at the same time, then stay in step. In addition, it would be nice to stop at some point, move to another, and pick up from there, with both machines still in sync. There are MIDI events that support such timing. Either of the machines, the sequencer or the drum machine, acts as a master. As the master runs, it sends these MIDI timing events to the other, the slave.

These timing events are:

START
Commands the device to start playing from the beginning.

STOP
Commands the device to stop playing.

CONTINUE
Commands the device to start playing from the current position.

CLOCK
Commands the device to increment its clock. This is the most critical part because it is the actual timing. The faster the CLOCK events are sent, the faster the music is played. The master keeps the slave in sync by sending CLOCK events at the rate being played.

SONG POSITION
Commands the device to set its clock to a particular time. If a CONTINUE command is sent next, it should start playing from there.

A good example of devices in SoundScape which produce and use MIDI timing events are the Clock and the Tape Deck (a sequencer). When the Start button is hit on the Clock module, it sends a START event, followed by a stream of CLOCK events, at some set rate. When the Tape Deck receives a START event, it rewinds to zero and enters the playback mode. From this point on, each time the Tape Deck receives a CLOCK event, it increments its internal clock counter and sends any notes that should be played.

The Clock module is a time base. With most sequencers, this feature is built in. Because of SoundScape's modularity, the two can be kept seperate. So, if a different time base, (such as a SMPTE reader or external MIDI) is desired, it can be connected and the Clock module can be disconnected, without the Tape Deck module needing to know about it.

With the Clock (or some future time base module) as our source, we can create modules that manipulate musical information, in sync with other modules that are also connected to the Clock.

continued...

MIDI Echo Module

A simple, but very useful module which needs timing information in order to process MIDI events is an Echo device.

To create an echo effect, take each note event that comes in and store it. After a set amount of time, send a copy of the note out, with the velocity damped a little. Wait the same amount of time and send another copy, with the velocity even lower. Repeat this process a set number of times, and you are done echoing this note.

In order to work, this module needs some timing reference. It could use the Amiga's Timer device as a source. The user would specify delay time in seconds. However, for most applications, it would be nice if the delay time were tied to the speed of the rest of the performance. The user could think in musical terms and set a quarter note as the delay time.

So, we use MIDI timing events for synchonization. This means that all the information we need to run (notes and timing) arrives as MIDI events, which streamlines things dramatically. We need to write one routine that processes the two different event types and install this routine as the 'outcode' (the routine each module provides which handles MIDI events sent to it.).

Here's a sketch of how this routine processes MIDI events:

We need four data items:

- *A list of all the notes that are currently being echoed - 'notelist'.*
- *A number that says how many MIDI clocks for the echo delay - 'delaytime'.*
- *A number that says how many times a note should echo - 'delaycount'.*
- *A flag that says whether we are running - 'running'.*

Here's how we process different events:

NOTEON, NOTEOFF:
If 'running' is true, and 'delaycount' is nonzero, store this event in 'notelist'. Since MIDI events are stored in the SoundScape Note data structure, we have two unused fields available: wait and duration. Store 'delaytime' in the duration field and 'delaycount' in the wait field. If 'running' is false, or there is no 'delaycount', simply return this event to the system.

START, CONTINUE:
Set 'running' to true.

STOP:
Clear 'running' and send all the events in 'notelist'. This is important so that notes which were on, will be turned off.

CLOCK:
If 'running' is true, go through 'notelist', decrementing the duration fields. For each event that has its duration (delaytime) at zero, do the following: Allocate a note event and copy this event into it. Set the velocity field of the copy to the original velocity * wait / 'delaycount'. This setting says that the velocity gets smaller for each successive echo. Send the copy. Set the duration field of the event in 'notelist' to 'delaytime'. Decrement the wait field (number of times left to echo.) If the number is zero, remove the event from 'notelist' and throw it away.

***All other events*:**
Return them to the SoundScape packet allocator.

Before we get to writing the code for this module, there is one issue that awaits us. We have two variables, 'delaycount' and 'delaytime', which we would like the user to be able to change. In addition, it would be nice if these could be loaded and saved as part of SoundScape environment load and save commands.

State Structures and Edit Routines

It would be nice if other modules could also edit, or at least access, certain parameters within your module. To do this, you define a data structure which has all the data about the state of your port. The first field is a long integer, specifying the length in bytes, of what follows. By noting the length, we make it possible for outsiders, that don't know anything about this module, to move the proper amount of data. This routine comes in handy for environment loads and saves. The remaining fields are whatever parameters you'd like to share with the world.

So, the appropriate state structure for the echo module would be:

```
struct EchoState {
    long length;
    long delaytime;
    long delaycount;
};
```

Then, to let the user and other programs access your variables, you must provide an edit routine of the form:

```
editcode(direction,command,state)
```

In this routine, `direction' specifies input or output, `command' indicates what type of edit operation, and `state' is a data structure which you have defined.

The commands are:

USEREDIT:
This command is called when the user double clicks on one of the port icons. `direction' specifies whether the clicked-on port was the input or output port. You should provide a user friendly edit routine. This means putting up a window with lots of Intuition gadgets to edit whatever parameters the user might need to get at. In some cases, you will need separate edit routines for the input and output ports. For example, though the Console Keyboard and Player Piano are the same port, they have separate edit windows. If your input and output are more tightly tied together, (which is most often the case), one window will do.

GETSTATE:
This commands you to copy your state structure into the buffer, including the length. If the caller doesn't know your state structure size, it will provide a buffer of 4096 bytes.

SETSTATE:
The opposite of GETSTATE, this command allows someone else to change your parameters. Copy the buffer into your state structure.

LOADSTATE:
This command is identical to SETSTATE, with one addition: After copying the buffer into your state, if your module stores something in files on disk (like samples), retrieve it. This is intended primarily for the load environment command. If you do file io, a file name probably should be part of your state structure. So, the LOADSTATE command ends up giving you the name of a file to load, which you copy into your state structure. Then load the file.

SAVESTATE:
This command instructs your port to save all files (if appropriate). Next, copy your state structure into the buffer. This is intended primarily for the save environment command; it will save your state structure. Later, when loading an environment, you can get that state structure back with a LOADSTATE command.

If you don't do file io, simply treat SAVESTATE as a GETSTATE command.

You may or may not have already run into one particular unpleasantry with AmigaDOS: Tasks can not do disk io. Well, you will run into that problem now, because your edit routine might often be called from a task. Doing disk io could be a problem.

SoundScape provides a way around this difficulty with a special function call that passes the address of your function and all the parameters to a process that does the io for you.

```
return = FunctionCall(yourfunction,parameter1,parameter2);
```


You are allowed up to seven parameters.

However, for our echo program, we don't do file io. So, the SAVESTATE and GETSTATE commands simply copy the EchoState into the supplied buffer, while the LOADSTATE and SETSTATE commands do the opposite.

We're ready to write our program.

(see listing for echo.c)

Here's the with file to link it:

```
FROM *
Lib:Astartup.obj,*
sslink.obj,*
echo.o,*
morelib.obj
TO *
EchoEcho
LIBRARY *
Lib:amiga.lib,*
Lib:lc.lib
```

Compile and run this program. Hook it up just as you did the chord program. Also, connect the clock to the echo module and start it running. Play notes. They should echo. Open the edit window by clicking twice on either icon. Change the parameters while playing notes. Do an environment save. Change the parameters again, then reload the environment. They should go back to the previous values.

With these two modules behind you, the possibilities are endless.

- *How about a module that takes notes and turns them into jumping frogs?*
- *Or your own sequencer.*
- *Or a module that computes harmonies to incoming notes.*

Have fun!

[Ed Note: Link files will be available through People Link and the Amicus disks.]

continued...

Listing for SoundScape.h

```
/*      SoundScape.H
        (c) 1986 Todor Fay
    Definitions and structure declarations for SoundScape.
*/

struct Link {
    struct Link *next;  /* Next node in the linked list. */
    unsigned char type;             /* Type of this node. */

        unsigned char mark;
        unsigned short data;
};

struct Note {
        struct Link link;
unsigned short duration; /* Clock beats the note is on. */
    unsigned short wait; /* Clock beats till next note. */
        unsigned char status;       /* Midi status. */
        unsigned char value;        /* Note value. */
        unsigned char velocity;
};

/*        Node types. */

#define NOTE       1

/*        Midi Commands */

#define NOTEOFF             0x80
#define NOTEON              0x90
#define POLYPRESSURE        0xA0
#define CONTROLCHANGE       0xB0
#define PROGRAMCHANGE       0xC0
#define AFTERTOUCH          0xD0
#define PITCHWHEEL          0xE0
#define SYSTEMX             0xF0
#define SONGPOSITION        0xF2
#define SONGSELECT          0xF3
#define TUNE                0xF6
#define EOX                 0xF7
#define CLOCK               0xF8
#define PUNCHIN             0xF9
#define START               0xFA
#define CONTINUE            0xFB
#define STOP                0xFC
#define PUNCHOUT            0xFD
#define ACTIVESENSE         0xFE

/*        Edit commands */

#define USEREDIT        1
#define GETSTATE        2
#define SETSTATE        3
#define LOADSTATE       4
#define SAVESTATE       5
```

Lising for Chord.c

```
#include "exec/types.h"
#include "exec/exec.h"
#include "intuition/intuition.h"
#include "soundscape.h"

/*First, the data for the icon in the Patch Panel  */

UWORD chorddata[] = {
0,0,
0,0,
0,0,
0,0,
0,0,
6,0,
1,32768,
255,57344,
1,32768,
6,0,
0,0,
0,0,
0,0,
0,0,
0,0,
0,0,
0,96,
0,96,
3072,96,
3072,992,
3072,992,
3072,96,
3072,96,
31744,992,
31744,992,
0,96,
0,96,
0,992,
0,992,
0,0,
0,0,
0,0,
};

struct Image chordimage = { 0,0,32,16,2,chorddata,3,0,0 };

/*This module has a port id. */

unsigned short thisport;

opencode(direction)

/*Always happy to open. */

unsigned char direction;

{
    return(1);
}

closecode(direction)

unsigned char direction;

{
    return(1);
}

outcode(event)

/*Strip the channel information from the status byte.  If
this is a NOTEON or NOTEOFF event, create two new events,
copy the status and velocity into them, add constants to
their note values, and ship off all three.  Otherwise,
free this event.
*/

struct Note *event;

{
    struct Note *secondevent;
    unsigned char status = event->status & 0xF0;
    if ((status == NOTEON) || (status == NOTEOFF)) {
        secondevent = (struct Note *) AllocNode(NOTE);
```

```
if (secondevent) {
    secondevent->status = event->status;
    secondevent->velocity = event->velocity;
    secondevent->value = event->value + 4;
    Send(thisport,secondevent);
}
secondevent = (struct Note *) AllocNode(NOTE);
if (secondevent) {
    secondevent->status = event->status;
    secondevent->velocity = event->velocity;
    secondevent->value = event->value + 7;
    Send(thisport,secondevent);
}
    Send(thisport,event);
    }
    else FreeNode(event);
}

long SoundScapeBase;

main() {
    SoundScapeBase = OpenLibrary("soundscape.library",0);
    if (SoundScapeBase) {
CloseLibrary(SoundScapeBase);
thisport =
AddMidiPort(opencode,closecode,0,outcode,&chordimage,
    &chordimage,-1,"chord maker");
SetTaskPri(FindTask(0),-20);
while (MidiPort(thisport)) Delay(100);
    }
}
```

Lising for Echo.c

```
#include "exec/types.h"
#include "exec/exec.h"
#include "intuition/intuition.h"
#include "soundscape.h"

/*This is the state structure for this module.  It has two
variables, delaytime and delaycount, that other modules can
access. The length field is always set to 8.  This indicates
that 8 bytes follow, so environment loads and saves, which
know nothing of this structure, can still move the proper
amount of data.
*/

struct EchoState {
    long length;
    long delaytime;
    long delaycount;
};

/*Initialise the delay time to 24 clocks (a quarter note) and
the delay count to 4 echoes.
*/

struct EchoState echostate = { sizeof(echostate) - 4,24,4 };
struct Note *notelist = 0;
unsigned short thisport;

UWORD echodata[] = {/* 44 x 10 */
0,0,0,
0,0,0,
24,0,0,
6,0,0,
255,32768,0,
6,0,0,
24,0,0,
0,0,0,
0,0,0,
0,0,0,
0,0,0,
6144,768,49200,
6144,768,49200,
6144,768,49200,
6144,768,49200,
6144,768,49200,
63488,7943,49648,
63488,7943,49648,
0,0,0,
0,0,0,
};

struct Image echoimage = { 0,0,44,10,2,echodata,3,0,0 };

/*Intuition gadgets for letting the user select delaycount
and delaytime.  These were generated with Power Windows:
*/


UBYTE SIBuffer6[16] =
"24";
struct StringInfo GadgetSI6 = {
SIBuffer6,
NULL,
0,
16,
0,
0,0,0,0,0,
0,
24,
NULL
};

USHORT BorderVectors1[] = {0,0,111,0,111,9,0,9,0,0};
struct Border Border1 = {
-2,-1,
1,0,JAM1,
5,
BorderVectors1,
NULL
};

struct IntuiText IText1 = {
2,0,JAM2,
-109,0,
NULL,
"Delay Time",
NULL
};

struct Gadget timegadget = {
NULL,
141,17,
108,8,
GADGHCOMP,
LONGINT+RELVERIFY+STRINGCENTER,
STRGADGET,
(APTR)&Border1,
NULL,
&IText1,
0,
(APTR)&GadgetSI6,
6,
NULL
};

UBYTE SIBuffer5[16] =
"4";
struct StringInfo GadgetSI5 = {
SIBuffer5,
NULL,
0,
16,
0,
```

continued...

```
0,0,0,0,0,
0,
4,
NULL
};

USHORT BorderVectors2[] = {0,0,111,0,111,9,0,9,0,0};
struct Border Border2 = {
-2,-1,
1,0,JAM1,
5,
BorderVectors2,
NULL
};

struct IntuiText IText2 = {
2,0,JAM2,
-109,0,
NULL,
"Delay Count",
NULL
};

struct Gadget countgadget = {
&timegadget,
141,32,
108,8,
GADGHCOMP,
LONGINT+RELVERIFY+STRINGCENTER,
STRGADGET,
(APTR)&Border2,
NULL,
&IText2,
0,
(APTR)&GadgetSI5,
5,
NULL
};


struct NewWindow NewWindowStructure = {
131,34,
286,51,
0,1,
GADGETUP+CLOSEWINDOW+RAWKEY,
WINDOWSIZING+WINDOWDRAG+WINDOWDEPTH+WINDOWCLOSE,
&countgadget,
NULL,
"Echo Parameters",
NULL,
NULL,
5,5,
640,200,
WBENCHSCREEN
};


opencode(direction)

/*Always happy to open. */

unsigned char direction;

{
    return(1);
}

closecode(direction)

/*When closing down, throw out notelist. */

unsigned char direction;

{
    struct Note *next;
    for (;notelist;notelist = next) {
next = (struct Note *) notelist->link.next;
notelist->velocity = 0;     /* Force to off event. */
Send(thisport,notelist);
    }
    return(1);
}


outcode(event)

struct Note *event;

{
    static char running = 0;
    struct Note *last, *note, *copy, *next;
    unsigned short velocity;
    switch (event->status) {
case START :
case CONTINUE :
    running = 1;
    break;
case STOP :
    running = 0;
    for (;notelist;notelist = next) {
next = (struct Note *) notelist->link.next;
notelist->velocity = 0;
Send(thisport,notelist);
    }
    break;
case CLOCK :
    if (!running) break;
    last = 0;
    for (note = notelist;note;note = next) {
note->duration--;
next = (struct Note *) note->link.next;
if (!note->duration) {
    copy = (struct Note *) AllocNode(NOTE);
    copy->status = note->status;
    copy->value = note->value;
    velocity = (note->velocity * note->wait) /
echostate.delaycount;
    copy->velocity = velocity;
    Send(thisport,copy);
    note->duration = echostate.delaytime;
    note->wait--;
    if (!note->wait) {
if (last) last->link.next = note->link.next;
else notelist = (struct Note *) note->link.next;
FreeNode(note);
    }
    else last = note;
}
else last = note;
    }
    break;
    }
    if (running && echostate.delaycount &&
      (((event->status & 0xF0) == NOTEON) ||
      ((event->status & 0xF0) == NOTEOFF))) {
event->link.next = (struct Link *) notelist;
notelist = event;
event->duration = echostate.delaytime;
event->wait = echostate.delaycount;
    }
    else FreeNode(event);
}

useredit()

/*Open a window with two string gadgets.  Wait for intuition
messages. If CLOSEWINDOW, close down and return. If RAWKEY,
send the keycode to OutConsole.  This is a SoundScape
routine that will pass the key value to the Console
Keyboard, so it can be played from this window.  If
GADGETUP, read the appropriate string gadget.
*/
```

```
{   struct IntuiMessage *message;
    short code, class;
    struct Gadget *gadget;
    struct Window *window;
    GadgetSI5.LongInt = echostate.delaycount;
    GadgetSI6.LongInt = echostate.delaytime;
    window = (struct Window *)
OpenWindow(&NewWindowStructure);
    for (;;) {
while (!(message = GetMsg(window->UserPort)))
    WaitPort(window->UserPort);
class = message->Class;
code = message->Code;
gadget = (struct Gadget *) message->IAddress;
ReplyMsg(message);
if (class == CLOSEWINDOW) break;
if (class == RAWKEY) OutConsole(code);
if (class == GADGETUP) {
    switch (gadget->GadgetID) {
case 5 :
    echostate.delaycount = GadgetSI5.LongInt;
    break;
case 6 :
    echostate.delaytime = GadgetSI6.LongInt;
    break;
    }
}
    }
    CloseWindow(window);
}
editcode(direction,command,copystate)
/*This is the edit routine.  If the command is USEREDIT, call
the routine useredit which puts up a window and gadgets and
lets the user edit the two variables.  The variable
notediting is used to avoid begin invoked multiple times. If
GETSTATE or SAVESTATE, copy the echostate into the supplied
buffer, copystate.  For SETSTATE and LOADSTATE, do the
reverse.
*/
char direction, command;
struct EchoState *copystate;
{
    static char notediting = 1;
    echostate.length = sizeof(echostate) - 4;
    switch (command) {
case USEREDIT :
    if (notediting) {
notediting = 0;
useredit();
notediting = 1;
    }
    break;
case GETSTATE :
case SAVESTATE :
    movmem(&echostate,copystate,sizeof(echostate));
    break;
case SETSTATE :
case LOADSTATE :
    movmem(copystate,&echostate,sizeof(echostate));
    break;
    }
}
long SoundScapeBase;
long IntuitionBase;
main() {
    SoundScapeBase = OpenLibrary("soundscape.library",0);
    if (SoundScapeBase) {
IntuitionBase = OpenLibrary("intuition.library",0);
CloseLibrary(SoundScapeBase);
thisport =
AddMidiPort(opencode,closecode,editcode,outcode,&echoimage,
    &echoimage,-1,"echo");
SetTaskPri(FindTask(0),-20);
while (MidiPort(thisport)) Delay(100);
CloseLibrary(IntuitionBase);
    }
}
```

•AC•

DATABASE
SPREADSHEET
WORD PROCESSOR

Micro-Systems Software —Bigger and Better

for the Amiga

NEW - ORGANIZE! DATABASE MANAGER, VERSION 1.0

Mailing lists! Club memberships! Patient records! Client files! Video tape libraries! Phone call logs! Nearly anything that needs to be filed, sorted or calculated is a candidate for **Organize!**

In seconds, **Organize!** can scan your files, locate information, and display or print in the format you want. Use it to print form letters with the Mailmerge function of Scribble!. Or calculate fields and do statistical analyses of your files with many of the same built-in math functions from Analyze!.

Easily design input forms and output reports with the mouse and pull-down menus. Just as simply - store, sort, review and print. The file size is limited only by disk space and the format is compatible with the industry standard dBASE format.

End your paper shuffle! Get **Organize!** today.

Only $99.95

UPGRADED - ANALYZE! SPREADSHEET, VERSION 2.0

ANALYZE! FEATURES:

- Pulldown menu interface (mouse-driven).
- Large spreadsheets with efficient memory usage.
- Dedicated function keys for common commands.

NEW ADDITIONS:

- **Business Graphics**; print bar, stacked bar, pie graphs in 2 or 3-D; line, X-Y, area graphs; all in 4 or 8 colors; data from spreadsheets; IFF format; view up to 4 graphs at same time; instantly redraw graphs when data changes; ranges, labels, titles, legends, rotation, scaling; fast and effective!
- **Command Macros**; save keystrokes; create templates.
- **Sorting**; rearrange row or column data quickly.
- File Icons; access spreadsheets via icons or names.

NEW PRICE Only $149.95

IMPROVED - SCRIBBLE! WORD PROCESSOR, VERSION 1.0

SCRIBBLE! FEATURES:

- Pulldown menu interface (mouse-driven)
- Multiple windows; edit/cut & paste 4 documents on screen.
- Preview; see final form on screen before printing.

NEW ADDITIONS:

- **Spellcheck**; expandable 40,000 word dictionary; check word, all words on screen, or entire document; alternative spellings shown.
- **Mailmerge**; print form letters, mailing labels; create data file with Scribble! or Organize!
- File Icons; access documents via icons or names; copy documents by pulling icons across workbench.
- Expanded Memory Support; for larger documents.
- More Amiga Keys; menu commands from keyboard or mouse.
- More Flexibility; Wordstar™ commands; scrolling while cut/paste; improved file operations.

Still only $99.95

MICRO-SYSTEMS SOFTWARE, INC.

4301-18 OAK CIRCLE, BOCA RATON, FL 33431
IN FL. CALL (305)391-5077 VISA, MASTERCARD

For Nearest Dealer Call
1-800-327-8724

See your local dealer or call:

Brown-Wagh Publishing

1-800-451-0900
1-408-395-3838 *(in California)*
16795 Lark Ave., Suite 210, Los Gatos, CA 95030

Wordstar is a trademark of Micropro International. Amiga is a registered trademark of Commodore-Amiga, Inc.

Waveform Workshop

in AmigaBASIC™

"A utility program for any AmigaBasic programmer who wishes to use the sound facilities of the Amiga."

by James Shields

Waveform Workshop is a utility program for any *AmigaBASIC™* programmer who wishes to use the sound facilities of the Amiga™. With Waveform Workshop, you can design, test, and save different waveforms, for use in other AmigaBasic programs.

I wrote Waveform Workshop because I needed to produce different sounding music in an AmigaBasic program. Consulting the available literature, I found that the prevailing opinion was that AmigaBasic programmers were limited to three basic sounds, and that only assembler or C programmers had access to the advanced music features. This didn't sound right to me. It had always been my impression that if you could change the wave shape, you could change the tone qualities of the note. I did some experimenting, and found that AmigaBasic could indeed produce many different types of sounds.

Sound is transmitted by pressure waves flowing through a medium, usually air. These alternating waves of high and low pressure move tiny membranes in your ears, which (by way of a few other bones and nerves) translate the pressure waves into the sensation of sound. Sound waves are usually periodic; that is, they repeat a particular pattern over and over. These patterns are called waveforms. Different musical instruments each produce unique waveforms. If you could save the waveform in the computer and play it back, the computer would sound like that instrument. The Amiga has the ability to recall and use a particular waveform in AmigaBasic. Thus the programmer can make the computer sound like any instrument he wishes.

So, what was needed was a way of editing and saving these different wave shapes for use in other programs. Given the way AmigaBasic produces sound (explained below), I designed Waveform Workshop to accomplish that task. I was so pleased with the results that I decided to share the program with others.

Sound and AmigaBasic

There are two fundamental AmigaBasic commands which affect the production of sound, WAVE and SOUND. The WAVE command is used to set the waveform pattern to be used by a particular voice of the Amiga. The SOUND command is used to tell the computer how and when to produce a note.

The format of the WAVE command is:

```
WAVE <channel>,<wave array>
```

The channel number, a number from 0-3, tells which of the four sound channels is to have the new waveform. The wave array is an array of integers which defines the shape of the waveform. The wave array must have at least 256 elements.

Possible values for individual elements in the wave array range from -128 to 127. If you graphed the values in the array, with the array index on the x axis and the values of the elements on the y axis, the resulting graph would show the shape of the wave.

Once the waveform has been described, the SOUND command is used to actually cause sounds to be made. There are three variants on the SOUND command. The most used variant is:

```
SOUND <frequency>,<duration>,<volume>,<voice>
```

Where *<volume>* and *<voice>* are optional.

Frequency is measured in Hertz and defines the note to be played. A higher value will result in a higher note being played. The range of values for frequency is 20 to 15000. Values outside this range will not result in an error, rather the computer will pick the closest valid note and use it instead.

Duration ranges from 0 to 77. A value of 18.2 is equivalent to a one second note.

Values for the volume parameter lie between 0 and 255. If this parameter is not specified, a value of 127 will be used. A value of 0 will produce no sound, whereas a value of 255 will result in an ear-splitting scream (well, as much as the Amiga is capable of...). Too much total volume (the sum of the volumes of all four channels) may result in sounding overtones -- excercise caution here.

continued...

The voice parameter is a value between 0 and 3, designating which channel is to play the note. If the channel is already playing a note, that note will be played out and the new note will be put on hold. Thus, a whole run of notes can be stored. One note of caution, however -- if too many notes are stored, AmigaBasic will run out of memory, and a nasty Guru Meditation will result. The actual number of notes that can be stored up depends on the size of your machine's memory, the size of your program, and several other factors. Thus I can't tell you how many can be stored. When I'm playing with this kind of thing, I try to be sure to save my program before I run it.

Other variations on the SOUND command include SOUND WAIT and SOUND RESUME. These commands stop and restart the production of sounds and are used to synchronize chords between multiple channels. The save memory caveat applies here. Be careful how many notes you play between a SOUND WAIT and a SOUND RESUME command, or you may find yourself out of memory, staring at a flashing orange box.

The Waveform Workshop

Listing 1 is the source listing for Waveform Workshop. I have tried to comment the code fairly well, so you know what is going on in a particular segment. Type teh codt in (with or without comments) and SAVE IT before you use it. RUN the program as usual. The screen will clear and the main display will appear.

Here is a list of the various functions and what they do:

Exit Terminates the program and returns you to the AmigaBasic environment. If you fail to save your waveform, you will still have a chance to do so here.

Play Plays a scale and some chords to demonstrate what the waveform sounds like..

New Deletes the current waveform and puts in the default value (a square wave).

Edit Allows you to change the current waveform. If you select this option, the bottom part of the screen will clear, and two new options will appear, "Compile wave" and "Exit edit". Now, just draw the shape of your waveform with the mouse. Press the left button to draw, and release it when you want to stop drawing. Go ahead and play with the wave until you like the shape.

If you decide you don't want to hear the wave, select "Exit edit" and you are through editing. If you do want to hear what the waveform sounds like, select "Compile wave". The computer will now digitize what you have drawn, and load it into memory. You then exit edit mode and you may play the demonstration. If you want to edit the wave again, select "Edit" and the whole process will begin again.

Display Draws out a series of waveforms at the bottom of the screen to show you what continuous stream would look like. Use this option to check how the ends of your waveform match up.

Name Allows you to give your waveform a name.

Save Allows you to save your waveform to disk. I usually use the save name as in the "Name" option. If you try to save a wave that is already on disk, the program will ask you if you really want to save it.

Load Allows you to load a previously saved waveform for further editing.

Noise Allows you to add a random "noise" element to your waveform. This will usually give your waveform more harmonics, resulting in a fuller sounding tone. You will be prompted for a percentage of noise, and the computer will generate a new waveform, based on the one you have designed. You will then hear a demonstration of the new wave, and you may accept or reject the noisy waveform.

Waveforms which have been saved by Waveform Workshop may be used in other AmigaBasic programs. Listing two is a sample routine for reading in a waveform file. You may read in as many waveforms as memory will permit, and you may use as many as four of them at any one time.

People who want to do complex music in AmigaBasic will find Waveform Workshop to be a valuable time-saving tool. Others will simply find the program fun to play around with! In any case, I feel the program will be a worthwile addition to your AmigaBasic library.

Listing One

```
'*************************************************************
'*                                                           *
'*            W A V E F O R M   W O R K S H O P              *
'*                                                           *
'*                      By James Shields                     *
'*                                                           *
'*                                                           *
'* Waveform Workshop allows the user to see and build        *
'* waveforms and save them as BASIC readable files, for      *
'* use in other programs.                                    *
'*                                                           *
'*************************************************************

Main:

   GOSUB constants      'Set up the constants and arrays
   GOSUB mainscreen     'Set up the main screen
   GOSUB waveedit       'Edit waves
   CLOSE                'clean up
   WINDOW CLOSE 1
   WINDOW 1,,,,-1
   END
```

```
'* Begin Subroutine *'

constants:     'set up program constants and arrays

   OPTION BASE 0
   DIM wav%(256),pat%(1)
   DIM savewave%(256),demo!(13)

' calculate note data for sound demo

   FOR i=0 TO 12
      demo!(i+1) = INT(263*((2^(i/12))))
   NEXT i
   wavename$="Noname"              'Name the wave
   filename$="No file"             'Tell where it came from
   pat%(0)=255
   pat%(1)=255                     'Set up pattern fill data
   FOR i%=1 TO 128
      wav%(i%-1)=127               'Set up initial wave data
      wav%(256-i%)=-127
   NEXT i%
   WAVE 0,wav%
   WAVE 1,wav%
   WAVE 2,wav%
   true% = (1=1)                'Symbolic boolean constants are
   false% = (1=0)               'used throughout.
   notsaved% = false%
   firstwave%=true%                'First time through

RETURN

'* Begin Subroutine *'

mainscreen:                        'main wave editing screen

SCREEN 1,640,200,3,2
WINDOW 1,"Waveform Workshop",,0,1
WINDOW OUTPUT 1

PALETTE 1,.6,.6,.6                 'define colors used
PALETTE 0,0,0,0
PALETTE 2,1,.9,0
PALETTE 3,.1,1,.3
PALETTE 1,1,1,1
PALETTE 5,1,.1,.1
PALETTE 6,.8,.13,.83
PALETTE 7,.27,.47,1

black  = 0              'Use symbolic names for colors
grey   = 1              'rather than color numbers.
yellow = 2              'Saves wear and tear on programmers'
green  = 3              'brains.
white  = 4
red    = 5
purple = 6
blue   = 7

CLS                             'Set the colors in use.
GOSUB mousereset

RETURN

'* Begin Subroutine *'

mousereset:     'wait until the mouse button is released

WHILE MOUSE(0) <>0
WEND
RETURN

'* Begin Subroutine *'

waveedit:                   'Set up the screen to edit
CLS
terminate% = false%         'We don't want to stop.
GOSUB wavescreen            'Display the screen.
```

continued...

```
WHILE NOT terminate%
LOCATE 1,1
PRINT SPACE$(65);           'Print the wave name and
origin.
LOCATE 1,3
COLOR purple,black
PRINT "Waveform:  ";wavename$;TAB(40);
PRINT "Filename:  ";filename$
COLOR blue,black
   waitformouse1:                          'get a command.
      IF MOUSE(0)=0 THEN waitformouse1
   x=MOUSE(1)
   y=MOUSE(2)
   IF (x<532) OR (x>545) THEN          'if there's an error
       PALETTE 0,1,1,1                 'flash the screen
       FOR i=1 TO 50
       NEXT i
       PALETTE 0,0,0,0
       WHILE MOUSE(0) <> 0
          WEND
       GOTO waitformouse1
   END IF

'Process the function selected.

   playwave%     = ((y>15) AND (y<23))
   newwave%      = ((y>31) AND (y<39))
   editwave%     = ((y>47) AND (y<55))
   displaywave%  = ((y>63) AND (y<71))
   namewave%     = ((y>79) AND (y<87))
   savewave%     = ((y>95) AND (y<103))
   loadwave%     = ((y>111) AND (y<119))
   noise%        = ((y>127) AND (y<135))
   exitwave%     = ((y>1) AND (y<9))

   IF playwave%    THEN GOSUB playwave
   IF newwave%     THEN GOSUB newwave
   IF editwave%    THEN GOSUB editwave
   IF displaywave% THEN GOSUB displaywave
   IF namewave%    THEN GOSUB namewave
   IF savewave%    THEN GOSUB savewave
   IF loadwave%    THEN GOSUB loadwave
   IF noise%       THEN GOSUB noise
   IF exitwave%    THEN GOSUB exitwave

   GOSUB mousereset

   WEND

firstwave%=false%
RETURN

'* Begin Subroutine *'

wavescreen:             'Print the main editing screen.

CLS
CALL box(533!,1!,544!,8!,grey)
LOCATE 1,70
COLOR blue,black
PRINT "Exit"
CALL box(9!,9!,523!,138!,yellow)
LINE (10,74)-(522,74),yellow
LOCATE 3,70
COLOR blue,black
PRINT "Play"
LOCATE 5,70
CALL box(533!,16!,544!,22!,grey)
PRINT "New "
CALL box(533!,32!,544!,38!,grey)
LOCATE 7,70
PRINT "Edit"
CALL box(533!,48!,544!,54!,grey)
LOCATE 9,70
PRINT "Display"
CALL box(533!,64!,544!,70!,grey)
LOCATE 11,70
PRINT "Name"
CALL box(533!,80!,544!,86!,grey)
LOCATE 13,70
PRINT "Save"
CALL box(533!,96!,544!,102!,grey)
LOCATE 15,70
CALL box(533!,112!,544!,118!,grey)
PRINT "Load"
CALL box(533!,128!,544!,134!,grey)
LOCATE 17,70
PRINT "Noise"

RETURN

'* Begin Subroutine *'

playwave:                   'Demonstrate the sound
GOSUB chords
playvave%=false%
RETURN

'* Begin Subroutine *'

chords:         'Play a scale and chords to demonstrate
SOUND RESUME    'the waveform sound.

SOUND demo!(1)/2,3    'c
SOUND demo!(3)/2,3    'd
SOUND demo!(5)/2,3    'e
SOUND demo!(6)/2,3    'f
SOUND demo!(8)/2,3    'g
SOUND demo!(10)/2,3   'a
SOUND demo!(12)/2,3   'b
SOUND demo!(1),3    'c
SOUND demo!(3),3    'd
SOUND demo!(5),3    'e
SOUND demo!(6),3    'f
SOUND demo!(8),3    'g
SOUND demo!(10),3   'a
SOUND demo!(12),3   'b
SOUND demo!(1)*2,3  'c
SOUND demo!(3)*2,3  'd
SOUND demo!(5)*2,3  'e
SOUND demo!(6)*2,3  'f
SOUND demo!(8)*2,3  'g
SOUND demo!(10)*2,3 'a
SOUND demo!(12)*2,3 'b
SOUND demo!(13)*2,3 'c1
SOUND WAIT                      'Syncronize the first chord.
'C
SOUND demo!(1)*2,20,140,0  'c
SOUND demo!(8),20,140,1    'g
SOUND demo!(1),20,140,2    'c

SOUND RESUME
'F
SOUND demo!(10),20,140,0   'a
SOUND demo!(6),20,140,1    'f
SOUND demo!(1),20,140,2    'c
'G
SOUND demo!(3),20,140,0    'd
SOUND demo!(8),20,140,1    'g
SOUND demo!(12),20,140,2   'b
'C
SOUND demo!(1)*2,20,140,0  'c
SOUND demo!(8),20,140,1    'g
SOUND demo!(1),20,140,2    'c

RETURN

'* Begin Subroutine *'

newwave:                    'Clear the old wave out
IF notsaved THEN GOSUB saveerror
GOSUB cleargraph
FOR i%=1 TO 128
```

```
      wav%(i%-1)=127
      wav%(256-i%)=-127
NEXT i%
wavename$="Noname"
filename$="No file"
notsaved = false%
newwave%=false%
RETURN

'* Begin Subroutine *'

editwave:                       'Actually edit the wave
GOSUB clearbottom               'Clear the dialogue window
COLOR yellow,black
LOCATE 23,5
PRINT "Compile wave";
CALL box(8!,176!,20!,182!,grey)
LOCATE 22,5
PRINT "Exit edit"
CALL box(8!,166!,20!,172!,grey)
GOSUB mousereset
mouseloop:                      'Draw what is pointed to.
IF MOUSE(0)=0 THEN      'If user released mouse button
   lastx=0              'Keep from drawing a bogus segment
   lasty=0
   GOTO mouseloop
END IF
x=MOUSE(1)
y=MOUSE(2)

'Check to see if a command was selected.

commandrange = ((x>7) AND (x<21))
IF (commandrange AND (y>165) AND (y<173)) THEN exitedit
IF (commandrange AND (y>175) AND (y<183)) THEN compilewave

'Check to see if the mouse is out of bounds.

IF (x<10) OR (x>522) OR (y<10) OR (y>137) THEN mouseloop

'If all is well, draw the segment of the wave.

IF lastx =0 THEN
   lastx=x
   lasty=y
END IF

'erase any segments in the same X plane as the new segment
IF lastx<=x THEN s=1 ELSE s= (-1)

COLOR black,black
AREA(lastx,10)
AREA STEP(ABS(lastx-x)*s,0)
AREA STEP(0,127)
AREA STEP(-(ABS(lastx-x))*s,0)
AREAFILL
LINE(lastx,74)-(x,74),yellow
LINE(lastx,lasty)-(x,y),red     'Draw the new segment.
lastx = x
lasty = y
GOTO mouseloop

'Wave editor commands:

compilewave:
   GOSUB recalcarray
   notsaved%=true%
RETURN

exitedit:
   editwave%=false%
   GOSUB clearbottom
   COLOR blue,black

RETURN

'* Begin Subroutine *'
```



```
recalcarray:            'read the screen data

GOSUB clearbottom
COLOR red,black

LOCATE 23,3
lastpoint = 10
wavedirection = 1  'is the wave rising or falling?

FOR i=0 TO 255
   j = lastpoint
   pointsscanned = 0
      scan:
      p=POINT(i*2+11,j)
      IF p=red THEN
         IF (j<lastpoint) THEN wavedirection=(-1)
         IF (j>lastpoint) THEN wavedirection=1
         ' and if its neither leave it alone.
         lastpoint = j
         wav%(i)=127-(2*(j-9))
         'bounds check
         IF wav%(i)>127 THEN wav%(i)=127
         IF wav%(i)<-128 THEN wav%(i)=-128
         LINE(i*2+11,j)-(i*2+11,j),blue
         GOTO nextpoint
      END IF
   j=j+(wavedirection)
   pointsscanned = pointsscanned+1
   IF (j<10) OR (j> 139) THEN
       wavedirection = wavedirection * (-1)
       j = lastpoint
```

continued...

```
   END IF
   IF pointsscanned<127 THEN GOTO scan
     'in case there is a blank space
nextpoint:
NEXT i

GOSUB redrawwave

'reset the waveforms for the demo

WAVE 0,wav%
WAVE 1,wav%
WAVE 2,wav%

'clean up and exit

notsaved% = true%
GOSUB clearbottom
COLOR blue,black
RETURN

'* Begin Subroutine *'

cleargraph:             'Clear the waveform graph

PATTERN ,pat%
AREA (10,10)
AREA STEP(512,0)
AREA STEP (0,128)
AREA STEP (-512,0)
COLOR black,black
AREAFILL
CALL box(9!,9!,523!,139!,yellow)
LINE(10,74)-(522,74),yellow
COLOR blue,black
RETURN

'* Begin Subroutine *'

displaywave:                'Display multiple waveforms
                            'at the bottom
                            'of the screen.
GOSUB clearbottom
LINE(1,158)-(639,158),yellow
FOR i=1 TO 256  STEP 2

'Write out the smaller waveforms 5 times (for speed's sake)

   LINE(i/2+1,159-wav%(i)/8)-(i/2+1,159-wav%(i)/8),red
   LINE(i/2+129,159-wav%(i)/8)-(i/2+129,159-wav%(i)/8),red
   LINE(i/2+257,159-wav%(i)/8)-(i/2+257,159-wav%(i)/8),red
   LINE(i/2+385,159-wav%(i)/8)-(i/2+385,159-wav%(i)/8),red
   LINE(i/2+513,159-wav%(i)/8)-(i/2+513,159-wav%(i)/8),red

NEXT i
COLOR blue,black
displaywave% = false%

RETURN

'* Begin Subroutine *'

redrawwave:                 'Draw the wave in wav%()

GOSUB cleargraph
lasty=wav%(0)
FOR i=1 TO 256
   LINE((i-1)*2+11,74-lasty/2)-(i*2+11,74-wav%(i-1)/2),red
   lasty=wav%(i-1)
NEXT i

RETURN

'* Begin Subroutine *'

clearbottom:            'Clear the dialogue window.

COLOR black,black
AREA (1,142)
AREA STEP (630,0)
AREA STEP(0,44)
AREA STEP(-630,0)
AREAFILL

RETURN

'* Begin Subroutine *'

namewave:               'Give the waveform a name.
GOSUB clearbottom
COLOR blue,black
LOCATE 22,3
INPUT "New name of wave";wavename$
namewave%=false%
GOSUB clearbottom
COLOR blue,black
RETURN

'* Begin Subroutine *'

savewave:                'Save the wave.
CLOSE #1        'Just in case
GOSUB clearbottom
COLOR green,black
LOCATE 22,3
INPUT "Filename (10 characters or less, EXIT to
quit)";filename$
IF filename$="EXIT" THEN exitsave
IF LEN(filename$)>10 THEN
    filename$=LEFT$(filename$,10)+".Wave"
ELSE
    filename$=filename$+".Wave"
END IF
ON ERROR GOTO newfile
```

```
'An error should occur if the file is not there.  If it is,
'then we go on and try to save.  Actually, an error here
'indicates that things are ok, and no error indicates things
'need to be checked out -- the wave already exists.

OPEN filename$ FOR INPUT AS #1
GOSUB clearbottom
COLOR red,black
LOCATE 22,3
PRINT "File exists; erase it? ";
GOSUB getyn
IF (answer$="n") OR (answer$="N") THEN savewave

newfile:
CLOSE #1       'Just in case it was open

OPEN filename$ FOR OUTPUT AS #1
FOR i=1 TO 256
   WRITE #1,wav%(i)
   LOCATE 23,3
   PRINT "Saving point ";i;
NEXT i
CLOSE #1

exitsave:
savewave%=false%
notsaved%=false%
GOSUB clearbottom
COLOR blue,black

RETURN

'* Begin Subroutine *'

loadwave:        'Load in a previously saved waveform.

loadwave%=false%
GOSUB clearbottom
COLOR blue,black
LOCATE 22,3
INPUT "Filename ";filename$
IF RIGHT$(filename$,5) <> ".Wave" THEN
       filename$=filename$+".Wave"
END IF
CLOSE #1       'Just in case
ON ERROR GOTO baddata

'Here, an error is really an error.

OPEN filename$ FOR INPUT AS #1
FOR i=0 TO 255
   INPUT #1,wav%(i)
   LOCATE 23,3
   PRINT "Reading point ";i;
NEXT i
CLOSE #1
wavename$=LEFT$(filename$,LEN(filename$)-5)
GOTO endload

baddata:
GOSUB clearbottom
COLOR red,black
LOCATE 22,3
PRINT "Unable to load file ";filename$;".  ";
IF ERR=53 THEN
   PRINT "File not found."
ELSE
   PRINT "File error."
END IF
PRINT "   Try again? ";
GOSUB getyn
IF answer$="y" OR answer$="Y" THEN GOTO loadwave

endload:
notsaved% = false%
GOSUB clearbottom
```

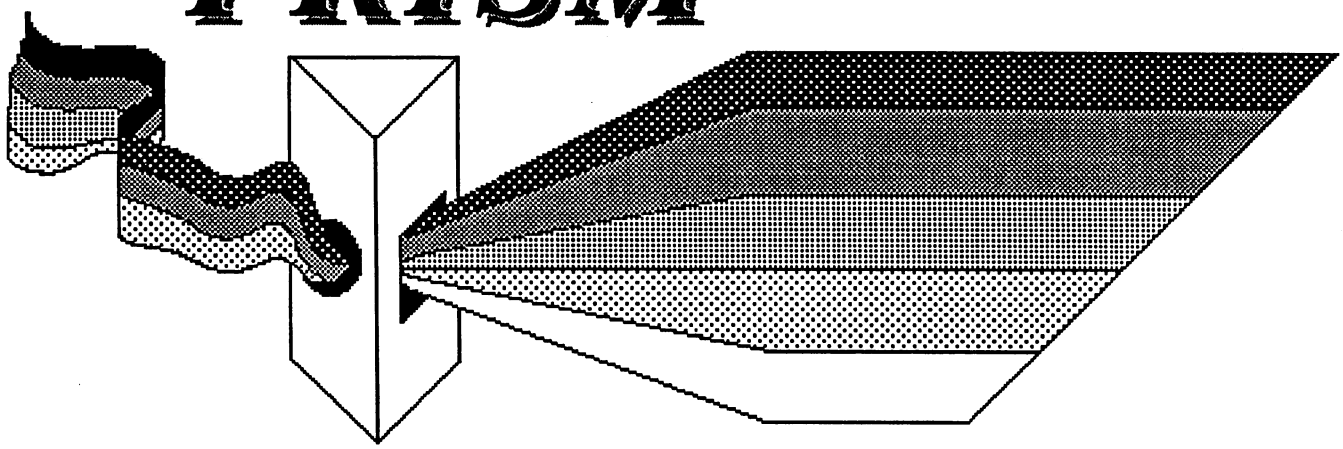


```
COLOR blue,black
GOSUB redrawwave

RETURN

'* Begin Subroutine *'

noise:       'Add noise to the waveform.


noise%=false%
checkloop:
GOSUB clearbottom
COLOR blue,black
LOCATE 22,3
INPUT "Percentage of noise";noiseamount
IF noiseamount>100 THEN GOTO checkloop
FOR i=1 TO 256
   savewave%(i)=wav%(i)    'Temporarily save the old wave.
   IF (RND*100)<noiseamount THEN wav%(i)=127-INT(RND*256)
NEXT i

GOSUB redrawwave    'Show the noisy wave.
WAVE 0,wav%
WAVE 1,wav%
WAVE 2,wav%

GOSUB chords        'See what it sounds like.
GOSUB clearbottom
COLOR blue,black

LOCATE 23,3
PRINT "Use this wave? ";
GOSUB getyn
```

continued...

PRISM

By Impulse



You have a computer that operates in a limited mode with most of the paint programs that are presently available. With *PRISM* you can operate in true *HAM* mode of 4096 colors. No longer are you limitied to 32 colors to paint with, now you have the full Amiga™ color pallete. Prism is designed to use the entire keyboard as well as the mouse to make painting and editing your artwork a real pleasure. Filp images, use a real lasso for grabbing objects and you can use any file that is IFF standard. Get the full power of your computer with *PRISM* see what you have been missing !!!

PRISM

only $69.95

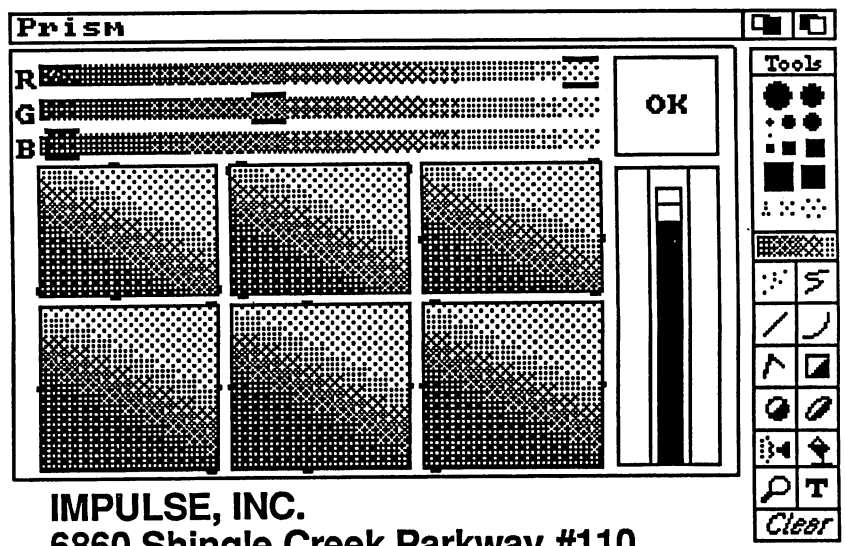
Here is the 4096 pallete screen. To pick your colors all you have to do is move your mouse and click that's all there is to using the power of the full 4096 pallete.

Don't delay, because now there are no limits on your creative talents. Go with your *IMPULSE* and get your copy of *PRISM*™ today.

TO ORDER CALL THE IMPULSE

HOTLINE

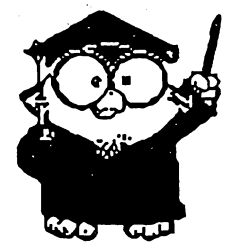
1-800-328-0184



IMPULSE, INC.
6860 Shingle Creek Parkway #110
Minneapolis, MN 55430

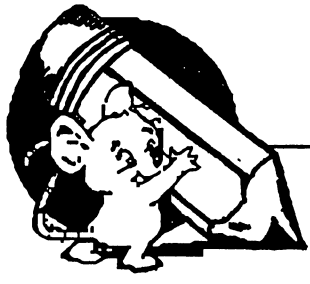
Go With The *IMPULSE*

ProClip



Artwork for

- DeskTop Publishing
- Sales Reports
- Greeting Cards
- Computer Video
- Animation



Now with PROCLIP you can produce documents that deliver your message!

5 PROCLIP DISKS

Over 200 images per disk!

- Space and Technology
- Leisure Time
- Sports
- Transportation
- Cartoons

Any one of these disks

Only $29.95 Each

TO ORDER CALL THE *IMPULSE* HOTLINE

1-800-328-0184

IMPULSE, INC.6860ShingleCreekParkway#110Mpls,MN55430

```
   IF (answer$="Y") OR (answer$="y") THEN exitnoise1
   FOR i=1 TO 256
      wav%(i)=savewave%(i)      'Restore the old wave
   NEXT i
   GOSUB clearbottom
   GOSUB redrawwave
   COLOR blue,black
   LOCATE 23,3
   PRINT "Try again? ";
   GOSUB getyn
   IF (answer$="n") OR (answer$="N") THEN exitnoise
GOTO loop3

exitnoise1:
notsaved% = true%

exitnoise:
GOSUB clearbottom
COLOR blue,black

RETURN


'* Begin Subroutine *'

exitwave:    'Finish up.

LOCATE 1,1
PRINT SPACE$(80);
LOCATE 1,2
COLOR red,black
PRINT "Exit wave selected."
IF notsaved% THEN GOSUB saveerror
terminate%=true%
exitwave%=false%
RETURN

'* Begin Subroutine *'
initwaves:              'Zero the waveform arrays.
FOR i =0 TO 255
   wav%(i)=0
NEXT i
RETURN

'* Begin Subroutine *'
saveerror:                      'Check to be sure the user
                                'wants to abandon the wave.
GOSUB clearbottom               'Clear the dialogue window.
LOCATE 22,3
PRINT "Changes made since last save.  Do you want to
save?";
GOSUB getyn
IF (answer$="y") OR (answer$="Y") THEN GOSUB savewave
RETURN

'* Begin Subroutine *'
getyn:                          'Get a yes or no reply.
ynloop:
answer$=INKEY$
IF answer$="" THEN ynloop
IF (answer$<>"Y") AND (answer$<>"N") AND (answer$<>"y") AND
(answer$<>"n") THEN ynloop
RETURN

SUB box (x1,y1,x2,y2,colr) STATIC        'Draw a box.
'Note that the numbers in the call must be long integers.
'constants must have a ! behind them, as in 511!.

LINE (x1,y1)-(x1,y2),colr
LINE (x1,y1)-(x2,y1),colr
LINE (x2,y1)-(x2,y2),colr
LINE (x2,y2)-(x1,y2),colr
END SUB
```

Listing Two

```
' demonstration routine for loading waveforms
' use the name of your file in filename$

DIM wav%(256)                       'integer array.
ON ERROR GOTO badfile               'if there's an error
CLOSE #1                            'just in case.
OPEN filename$ FOR INPUT AS #1
FOR i=1 TO 256                      'get 256 elements.
   INPUT #1,x
   wav%(i)=x
NEXT x

badfile:

'You can insert any error handlers here

CLOSE #1
IF ERR<>0 THEN PRINT "Error ";ERR;" in waveform read."
```

•AC•

AmigaNotes

Mimetics SoundScape Sound Sampler

by Richard Rae
CIS# 76703,4253

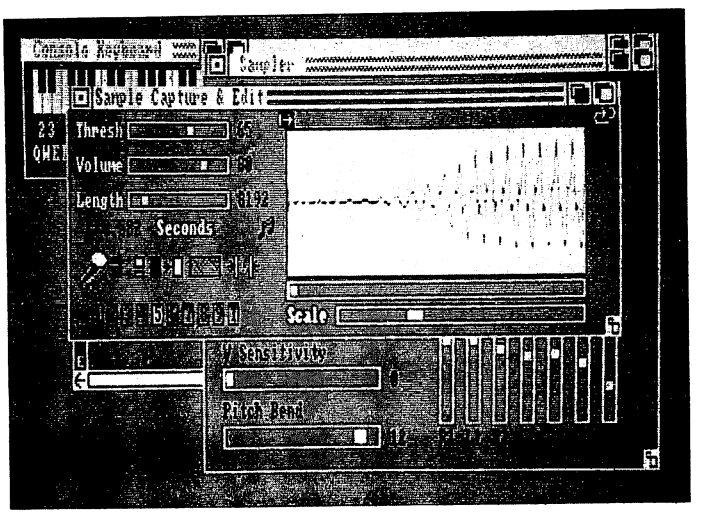
The SoundScape audio digitizer is the smallest unit currently available. It is housed in a black metal box which plugs into the second mouse port. Near the bottom of the unit are two RCA jacks for line level inputs, and a miniature jack for microphone level signals. The unit is shipped with a 28 page manual, disk, and warranty card, all (except the sampler) enclosed in a very nice vinyl slipcover. The warranty period is 90 days.

The sampler disk can be used from either WorkBench or CLI, and is not copy-protected. Interestingly, although the software is on a bootable disk, there are no provisions for loading WorkBench. If you wish to use the icon driven environment, you must either boot from another disk or copy LoadWB to the SoundScape disk.

There are several files on the disk; the main program is called Sampler. Running this program brings up two windows: a tiny Console Keyboard window and the main window. The Console Keyboard is a visual aid which shows you which keys on the keyboard corresponds to the keys on a musical keyboard. If the window is active, it will also show which of these keys are depressed. By using the console keyboard you can play up to four notes simultaneously, making this the only sampling package which is a complete, stand-alone musical instrument. In addition, the SoundScape system responds to MIDI input on channel 1 as a velocity sensitive four voice instrument.

Within the main window there are conceptually three groups of controls. The right half of the screen is devoted to the envelope and looping of the sample. To the left, is the "performance" group: sliders and gadgets which control how the samples react to the controlling device. Above the performance group, in the upper left hand corner, are gadgets used to manage samples. Let's take a look at each of these.

A sample may be either a one shot affair such as a marimba or snare drum, or continuous, like an organ. Rather than wasting memory by recording long continuous samples, we can instead **loop** a shorter sample to stretch it out in time. The SoundScape package provides sliders for setting the start of the sample, the start of the loop, and the end of the loop. The sliders on the main screen are remnants of an



earlier version, and provide coarse adjustments. These sliders are useful for making quick experimental changes. For more precise work, you should use the sliders in the Sample Capture and Edit window.

The envelope controls allow you to set the amplitude contour of the sample. In other words, you can define how the volume of each note changes over time. Four rate and three level sliders are supported, providing somewhat more control than a traditional ADSR envelope generator. Note that standard IFF format does not have provisions for envelope parameters of this complexity; you must save samples in Mimetics' own format to preserve this data.

The "performance" group is comprised of controls which alter the sample's response during playback. A High/Low gadget above the performance sliders allows you to either raise or lower the sample's frequency by one octave; the direction in which you can change the pitch depends on the setting of this gadget when the sample was recorded. The Tune slider allows you to detune the sample from its recorded pitch for use with a recording or other instrument. This tuning is done with one cent resolution (one cent is 1/100th of a semitone). Although both the manual and screen indicate that the tuning range is plus or minus 100 cents, the actual range is somewhat less than this.

continued...

The Transpose and Octave sliders allow you to alter the key of the recorded sample. Using the Transpose slider, you may raise the pitch in half-step increments up to 11 steps. Similarly, the Octave slider will raise or lower the playback range by up to four octaves. Unlike the Tune slider and High/Low gadget, which actually change the pitch of the sample, these controls simply slide the sample up and down on the keyboard. For example, if you have a flute sample whose lowest note is A, and you transpose it down three semitones, you will hear the A when you play a C, but the keys below the C will be inactive.

The V Sensitivity slider controls the response of the system to velocity information. This data reveals exactly how quickly (and, indirectly, how hard) you hit each key, and is transmitted by some MIDI instruments. In the SoundScape system, this velocity data is used to control the volume of the sample.

The Pitch Bend slider, like V Sensitivity, is used only with external MIDI input; It determines the sensitivity to the controlling instrument's pitchbend wheel. This slider spans a range of 0 to 12 semitones, so the Amiga can be set to ignore the pitchbend wheel or bend up to plus or minus one octave.

The section I refer to as "sample management" includes gadgets to load and save samples, which bring up standard directory and file name requestors. When saving a sample, you have the option of using standard IFF format or Mimetics' private format. The latter will take more disk space but will save **all** sample information, including envelopes and performance settings. The Clear gadget erases the current sample from memory, freeing up space for further work. Finally, the Sample gadget brings up the Sample Capture and Edit window. And thereby hangs a tale...

I received one of the first SoundScape samplers quite some time ago, and have been holding on to it with the intention of doing this series of reviews. A few months ago, I received a software update which has changed my opinion of this package from "not ready" to "very nice". The new version appears to be nearly identical to the earlier release, until you open the sample window. Just a brief period spent working with the updated version will convince you that Mimetics has a very usable system.

SAMPLE CAPTURE AND EDIT

Mimetics provides three sliders to control the sampling procedure. The Threshold slider sets the level at which recording will automatically begin. By setting this slider to zero, you can manually start capturing the sample. At higher levels, the software will wait for the input to exceed the preset level. The former mode can be used for clipping a sample out of a prerecorded song, whereas the latter mode is handy for recording a natural sound such as breaking glass.

The Length slider determines how much memory is to be used for the sample. This slider can be set from zero (in which case recording is disabled) to 65534 bytes, which results in the maximum 4.66 second sample time. In the previous version, the limit was 32766 bytes (yes, really!), which means your samples can now be twice as long. Be aware, however, that some programs will have problems with IFF samples longer than about 32K. Longer samples are most useful in Mimetics' format.

The default sample rate of 15KHz is probably the best compromise for sampling instruments: raising the sample rate won't do much for you in terms of sound quality because of the Amiga's output filters, and lowering the rate too much is going to allow aliasing noise into the samples. Still, for games and video presentations which need sound effects, there will be times when programmers will want to trade sound quality for less memory usage and faster loading of samples. The new version of the software gives us that capability. The High/Low gadget in the main window originally controlled playback rate only. Now it also controls the record sampling rate. The gadget defaults to High, but by clicking it to Low before opening the sample window you can record at a 7.5KHz rate. This change provides up to 9.32 seconds of recording time in the same 64K of memory, at the expense of sound quality.

Perhaps the most important change for many of us is the addition of the Volume slider. Since there is no level controls on the sampler itself, users of the first release were at the mercy of whatever signal they happened to feed the hardware. A signal which was too low in volume resulted in a poor signal-to-noise ratio, while too high a level resulted in very obnoxious distortion from clipping. With the new Volume slider, the user can now set the gain of the sampler to match the signal source.

The ability to change gain with software is a byproduct of the sampling method used, which differs from that used by other Amiga samplers. This particular approach to sampling also allows Mimetics to include a compression gadget which, when engaged, attempts to maintain a high overall level during recording. The effect of this compression can range from insignificant to remarkable, depending on the sample in question.

To support the Volume slider, the sampling operation itself was also changed. Previously, clicking on the Microphone icon froze the cursor and waited for the input signal to exceed the value set by the Threshold slider. At that time, the screen went blank (to the background color) while the recording occured.

All that has changed. Now, as soon as the Microphone icon is clicked, the screen goes solid red ("Wait") and the audio being fed to the sampler is routed, in real time, to the Amiga's audio outputs. Now you can hear what you are sampling. During the red screen (which lasts less than a second), the software adjusts the gain of the sampler to match the setting of the volume slider. Once this is done, the screen goes solid yellow ("Ready"). At this time, the software is waiting for your signal to go ahead with the acquisition process.

Clicking the left mouse button once, while the yellow screen is up, will cause the software to watch for a value exceeding the Threshold setting, just as in the previous version. When that signal is detected, the screen goes green ("Recording") and storage begins. As soon as the buffer set by the Length slider is filled, the normal screen returns.

Another change in software operation is an "escape hatch". Previously, if you set the threshold too high for your input signal, there was no graceful way to exit the "wait" condition. With the current software, you need only click the left mouse button a second time on the yellow screen and recording will start immediately.

SoundScape provides ten octave gadgets in which you may store samples. You can sample directly into any of the ten by clicking on the appropriate gadget before beginning your recording. (A temporary work area is also provided for sample shuffling.) Once a sample is in place, you can copy it directly to another octave for manipulation, or you can actually raise or lower it to match the target octave during the move. The latter operation is performed by either clipping out or doubling points in the sample. Some caution is required when attempting to fill a large number of octaves with one sample. Each time you lower the octave you double the storage required for the new sample, and you can quickly find yourself running out of workspace. If you are sampling an instrument, it is much better to actually sample each octave separately, rather than attempting to shift one octave in this manner. (This is true for a number of other reasons as well.)

Actually, the term "octave" is somewhat misleading, as the various workspaces do not have to be octaves of the same instrument. There is nothing to prevent the user from multi-sampling; That is placing a different instrument or sound in each workspace. The drum kit used by Instant Music is an excellent example of what can be done in this manner.

Perhaps the most exciting change since the last release is the addition of Visual Editing, allowing the user to directly view and manipulate the waveform as it is stored in memory. The right half of the sample window is occupied by a waveform display area and associated sliders. When a sample is loaded into the system, the actual waveform is drawn into this area. The Scale slider beneath the display allows you to determine how much of the sample you will see, and at what resolution. With this slider all the way to the right, you see the sample in its entirety; fully to the left you can see each individual point of a very tiny slice. Above the Scale slider is a positioning slider, which allows you to move the waveform around in the display area so you can see any portion, at any magnification. Above the display window are three sliders which represent the start, loop start, and loop end points. Their positions accurately reflect these points in the waveform.

All this is nothing new; other Amiga samplers do the same thing. The difference here is that you can **change** the sample. With the Scale slider near the left end, you can actually use the mouse to **draw** waveforms into the display area. This is an incredibly handy tool that "real" sampling systems have had for a while, and I'm delighted to see appear on the Amiga so soon.

What can you do with this ability? Well, if you want to just tinker, you can try drawing waveforms into the display area freehand to see what they sound like. In fact, you can do this in real-time by using a little trick. Start by moving the Scale gadget all the way to the left. Next, set the loop sliders to either end of the display area. Now, depress the "Y" or any other active key and, while holding it down, move the mouse into the display area and push the left mouse button. Then, release the key while continuing to hold the mouse button. Because of the way the software is written, the program will not see the release of the key because it is watching the mouse button. As far as the program is concerned, you are still holding down the key, which means the note will play continuously. Now, you can tinker with waveforms as long as you like, with instant gratification. When you find something you like, just touch and release the same key again to turn off the sound.

Freehand drawing is interesting, but there are other applications -- not the least of which is repairing a damaged waveform. Suppose you have sampled a one time event -- an absolutely wonderful cat yowl, for example -- and it is perfect except for the volume being a bit high, resulting in distortion at one point. With practice, it is possible to

continued...

actually remove this distortion from the sample! It isn't easy, since it takes practice and a steady hand. You'd best save a copy before you start editing... once you've made a change, it is forever. But, with care and patience you can repair a lot of problems. I've also had great success removing a "pop" from a sample taken from a record.

OTHER GOODIES

As mentioned, Sampler is the main driver program. Also included is RAMSample, a demonstration program which can use all available memory for a single sample. RAMSample is keyboard driven and allows you to record into memory with or without monitoring the signal at the Amiga's output. This sample can then be played back once or continuously. Provisions are also made for saving to and loading from disk.

The folks at Mimetics included an extra program on my disk: a little demo routine called Stereocompr. When the SoundScape digitizer was first released, many owners complained that, although it was advertised as capable of stereo, the digitizer it actually wasn't. In fact it simply summed both channels together and recorded in mono. This is not the case now. The hardware is indeed capable of stereo, and it is the **software** which is doing the summing. Stereocompr is a stereo in, stereo out audio compressor which pipes audio through the SoundScape digitizer and my Amiga. Although useless except as a demonstration program, Stereocompr does prove conclusively that the hardware is capable of stereo input. This program is a hint of good things to come, and Mimetics maintains that even the software shipping now only makes use of a fraction of the sampler's capabilities. They also indicate that owners who have sent in their warranty cards will automatically receive updates as new versions are released.

The digitizer disk also includes 14 instrument samples and one RAM sample. Those of you with the old dealer Instruments demo will find these somewhat familiar.

IRRITATIONS

Yep. You know by now that it wouldn't be a true AmigaNotes review if I couldn't find something I didn't like.

The lack of a feedthrough port is going to be a problem for many of us. I know people who have joysticks, TIC real-time clocks, and copy protection dongles, all of which have to be plugged into the second mouse port. Couple this difficulty with the manual's instructions to turn the Amiga off before inserting or removing the sampler, and you have a rather inconvenient situation.

The input jacks on the sampler are down near the bottom of the box, whereas it plugs into the mouse port near the top. What this means is that plugging or unplugging an audio cable tends to lever the sampler right out of the mouse port. To avoid this, you must support the sampler with your other hand somewhat awkwardly.

I wish Mimetics had implemented the volume control exactly opposite from the way they did. When you click the microphone icon and the screen goes red, the sampler comes up at full gain and is turned down by the software to match the level set with the Volume slider. Unfortunately, this usually means you are treated to a blast of loud, distorted sound from the Amiga's outputs until the software gets everything set up properly. Had they instead, started at minimum gain and ramped up, this would have been avoided.

Another problem with the Volume slider is the gyrations you must go through to properly set it. Ideally, you should be able to monitor the audio at the Amiga's outputs while adjusting the Volume slider for maximum level without clipping. But you can't. The best you can do is set the Length slider to a very small value, guess at a Volume slider setting, click the microphone icon, wait for the yellow screen to listen to the results, double click the mouse to go through a dummy recording cycle and return to the sample screen, and try another Volume slider setting.

The only real bug I found is related to the envelope controls. When the L3 slider (which controls the sustain level of the note) is set to maximum, the R4 slider (which controls the final release rate) becomes inactive. When a key is released, the note stops abruptly, regardless of the setting of R4. This can be circumvented by pulling L3 back slightly.

Congratulations!

from

AmiEXPO, the Amiga™ specific conference and exhibition, applauds all those who in the past two years had the foresight and knowledge to purchase Amiga computers.

In recognition, we extend a special invitation to join a dynamic new event:

AmiEXPO.

Where & When

Fall

Sheraton Centre Hotel
October 10-12, 1987
New York, New York

Winter

Airport Hilton Hotel
January 22-24, 1988
Los Angeles, California

Spring

Hyatt Regency Hotel
July 22-24, 1988
Chicago, Illinois

AmiEXPO will feature a full range of exhibitions, conferences and seminars, all designed to meet the challenges of the Amiga community. By combining all of these forces, **AmiEXPO** will provide a unique forum for discussion and trade, in a conducive environment.

If you would like to register or exhibit, call **800-32-AMIGA** (in New York State call 212-867-4663) or complete the form below and return it to:

AmiEXPO Headquarters.

YES, send me more info on **AmiEXPO!**

Name: _____
Company: _____
Address: _____
_____
_____
Telephone: _____

Return form to:
AmiEXPO Headquarters
211 East 43rd Street, Suite 301
New York, New York 10017

Amiga™ is a registered trademark of Commodore-Amiga,Inc.

In addition to this bug, I found a few of what I would consider "bugettes"... not true bugs, but quirks that certainly shouldn't be there. For example, when the sample window is on screen, you cannot access the main window, even if that window is shown as active. You can click in the main window and the sample window will ghost. You can even grab sliders in the main window and slide them back and forth. But the values will not change unless you close the sample window and THEN change the sliders. This is inconvenient, since the Octave slider needed to access various samples is on the main screen.

Some operations do not properly refresh the screen. For example, the default Octave slider setting is 0. If you set the slider to -1 and then load a Mimetics format sample which was saved at octave 0, the 0 overwrites the minus sign, and you are left with "01", which is easily misinterpreted.

If you click the Load icon after saving new samples to a disk, and had previously loaded from that disk, SoundScape will present you with the old listing of samples. You will not see the new samples until you change disks or at least put the cursor on the drive specifier and hit return.

IN CONCLUSION

All things considered, the SoundScape digitizer is a good value. It appears to be geared primarily towards creating sampled instruments, and therefore, leaves out features like continuously variable sample rate and extremely long samples. These features would be handy for recordings intended for use in other programs. On the other hand, SoundScape is the only sampler which gives you everything you need to play sampled instruments directly from your computer keyboard OR a MIDI instrument. The SoundScape digitizer software also plugs into the Pro MIDI Studio as a module, minimizing the hassle involved with getting samples into a Pro MIDI score... nice.

That's it for now.

Nybbles,
Rick

•AC•

SoundScape Digitizer $99.00
No Copy Protection
Requires Amiga with 512K, one drive, KS 1.1 or up.

Mimetics Corporation
PO Box 60238, Station A
Palo Alto, CA 94306
408-741-0117

More AmigaNotes

by Richard Rae

SunRize Industries' Perfect Sound Audio Digitizer

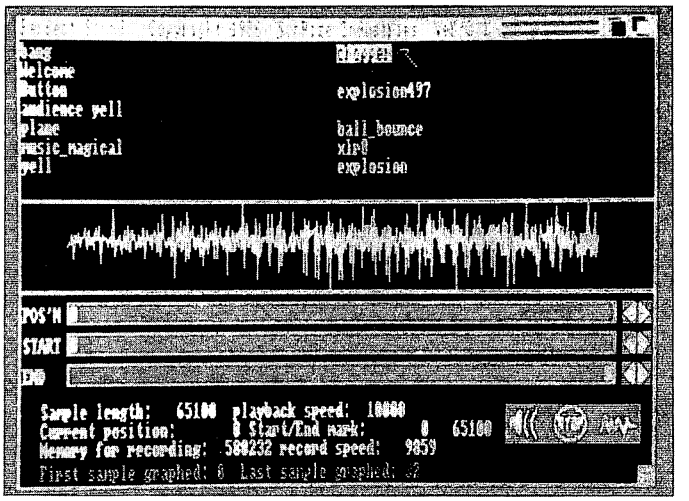
"....the only sampler which is capable of stereo sampling, right out of the box."

SunRize has chosen an interesting route for marketing their Perfect Sound digitizer. The sound editor software has been declared copyrighted shareware and distributed to bulletin boards and commercial networks across the country. The shareware package is completely usable without the hardware, and even comes with demo samples. A person with a different brand of sampler, or no sampler at all, can still use the Perfect Sound editor to modify their samples. A contribution of $20 is requested if you find the software useful, or you can buy the complete sampler package, in which case the software is included. This approach allows you to "try before you buy", at least as far as the software is concerned, plus gives people who already own a sampler another type of editor to work with.

The Perfect Sound digitizer is a thin, cream colored box which plugs into the parallel printer port and projects up over the back of the Amiga. Two RCA jacks are provided on the right side for line level inputs (you'll have to preamplify microphone level signals), and two forward facing controls adjust input gain. Along with the digitizer, you'll also receive a 20 page manual, disk, registration card, and a couple of disk labels, all shrink wrapped into a tidy little package. The warranty period is 90 days.

SunRize is another company which appears to be willing to improve their products whenever they can. The first Perfect Sound digitizer I received was epoxied between sheets of gray plastic. There was no case per se; you could look in the end and see the components. The digitizer plugged into the parallel port horizontally, meaning you had to pull your Amiga about five inches away from the wall to accomodate it. And, instead of real controls, it had two tiny trimmer potentiometers mounted on the back side, which turned in opposite directions relative to each other! The digitizer currently shipping is a vast improvement over their first model, and I compliment them on the changes they have made.

This digitizer still seems to lack some of the "flash" of the others available; it's simply not as "clean" looking. And the manual is fairly superficial, providing little in the way of examples. On the other hand, this is the least expensive of the three units currently shipping, and it is the only sampler which is capable of stereo sampling, right out of the box. For those advantages a lot of us might choose to give up a little glamour!



THE SOUND EDITOR

The main sound editor program is called PSound, and can be called up either from WorkBench or the CLI. If started via its icon, or from the CLI when logged into its directory, PSound will present an opening screen and brief demonstration before dropping into the editor. You can avoid this, and go directly to the editor, by using CLI from another directory.

The Perfect Sound editor divides the screen horizontally into two sections. The larger upper section is devoted to "slots" for the various samples you will be working with. The editor can deal with up to 14 samples simultaneously, and you select the active sample by clicking its name once with the mouse; this highlights the name. Clicking on a selected sample will play it in its entirety.

The bottom third of the screen contains two pushbutton gadgets, three sliders, and a status display area. The "Play Sample" gadget performs the same job as clicking the name of the selected sample. "Play Range" plays the portion of the sample between the Start and End sliders. The third slider, Position, is used to indicate the insertion point for a sample; I'll touch on this a bit later.

The status area constantly displays the amount of memory remaining for samples, the length of the selected sample, the start and end locations, its record and playback rates, and a status line for editor messages.

continued...

There are five menus accessable via the menu bar: Edit, Special, Digitize, File, and the traditional "About". Let's take a look at each in turn.

The Edit menu highlights the fact that this package is intended for building specialized samples, rather than simply recording sounds and creating instruments. There are quite a few unique functions here which could come in handy when building a complex composite sample from individual snippets.

Generally, the way you would approach making a sample is to start recording a bit early, stop a tad late, and then carefully snip off the excess from each end to leave exactly what you want. "Delete Marked Range" allows you to do this, and more: the marked range could also be in the middle of the sample. Lots of interesting uses come to mind for this one... for example, assume you are writing a game which includes a jury trial, and you have a sample of a juror saying "Your honor, we find the defendant not guilty". By snipping out the word "not", you can completely reverse the jury's decision. Depending on the background noise and inflection of the speaker, it is possible to do this so cleanly that it can't be detected.

"Copy Range To New Slot" allows you to pick out a piece of a sample using the Start and End sliders, and then copy this segment to a new slot. In addition to providing a simple method for extracting a sound from a longer sample, it is a safer method of trimming than deleting part of a sample: the original sound remains intact in case you make a mistake.

"Append Slot To Slot" tacks one sample onto the end of another sample. This is the most straightforward way to build a composite sample: clip out the individual sounds you need, placing one in each slot, then append the sounds together, in order, to create the final sample.

"Insert Marked Range" is another approach to building complex samples. With this option, you can actually mark a portion of a sample using the Start and End sliders, then "plug it into" another sample. The Position slider is used to indicate where you wish to insert the addition.

"Create Stereo" is one way to build a stereo sample: it combines the two samples you indicate into a new sample, with one selection on each output channel. This could be handy for layering gunshots on top of squealing tires! Be careful with this one, though... rather than building a third sample from the original two, it combines both into one new stereo sample under the name of the first sample. If you need to keep the original samples, you'll have to make copies first.

"Break Up Stereo" is the inverse of "Create Stereo": it breaks a stereo sample into left and right components. You'll use this more than you might expect, because several operations will not accept stereo samples.

"Change Playback Period" brings up a requester with a slider for setting the playback rate. The range is from 124 to 700, with the selected rate shown in a small box below the slider. By clicking in this box, you can enter any value directly, although the Amiga will not use a playback period smaller than 124. Be advised: don't try ridiculously large numbers or zero, unless you are willing to wait a very long time (or reboot).

"Discard Sample" deletes the selected sample from memory, opening that workspace for a newer project.

The Special menu contains a variety of interesting commands. "Flip" actually reverses a sample's direction, causing it to play backwards. "Freq=Freq*2" raises a sample's frequency an octave by snipping out every other point in the sample. Similarly, "Freq=Freq/2" drops a sample an octave by doubling every point.

"Graph Marked Range" uses a narrow strip across the center of the screen to draw the sample between the Start and End positions. When this menu item is selected, the status line will show the value of the first and last points. This is handy for butting samples together to make composites, or for creating instruments, since any sudden change in the waveform will result in a click. Other than for this and checking recording levels, I didn't find the graphing function terribly useful. It does **look** impressive, though!

"Auto Graph" redraws the graph every time any change is made to the sample, which includes any movement of the Start and End sliders. Since it takes over two seconds to redraw the graph, this option tends to slow down the work quite a bit, and should be used judiciously.

"Create Instrument" is used when you wish to create a standard IFF instrument rather than a specialized sound effect. Although it's nice that SunRize provided this capability, it can be somewhat difficult to use. You must clear out all samples except the ones to be used to make the instrument, and you must be sure to arrange them in descending octave order (unless you are building a multi-sampled drum kit, for example). The editor will warn you if any of your octaves are not exact multiples, or if your repeat length is uneven, or if a sample does not begin on a word boundry, but it does not assist you in correcting these conditions. Once you have created an instrument, you can listen to one note from each octave by using the function keys, but it is not possible to hear any other notes from the scale.

The Digitize menu contains all the options related to recording a sample. "View Signal Levels" calls forth a continuous display of minimum, maximum, and current levels for both the left and right channels, as well as the difference between them. This display remains active until the left mouse button is clicked inside the window.

"Alter Record Speed" works in the same way as "Change Playback Period", and controls the sampling rate during actual recording. The slider range is from 124 to 999, which is roughly a range of 28,800 to 3583 samples per second. (If you've always wondered how these sampling periods relate to the real world, you divide 3579545 -- the color burst frequency -- by the sampling period to get samples per second.)

"Monitor Digitizer" takes over the computer and pipes the input from the digitizer to the audio outputs of the Amiga in real time. You can monitor either channel, or both together in stereo. This is an excellent way to set digitizer gain, since you can hear any distortion. Unfortunately, the monitoring occurs at a fixed sample rate. Had it been tied to the selected record speed, it would have also been useful as a preview of exactly how the final sample would sound.

"Record Sample" allows you to capture a sample from either input, or in stereo. The system enters "Monitor Digitizer" mode and waits for you to click the left mouse button, at which point the monitoring ceases and recording begins. Recording will continue until you click the left button again or you run out of sample memory. When sampling is completed, you are asked for a name for the sample, and it is added to the editor's list as the currently selected sample.

The File menu allows you to load or save a sample in any one of three formats: IFF, DUMP, and COMP. IFF is the standard interchange format among Amiga music and sound programs. DUMP is a raw dump of the sample. It will save a few bytes and will load a bit faster than IFF, but is not usable by all music programs. (This option is useful if you are writing a program which needs sound, but you don't care about or need IFF routines.) COMP is a variant of IFF which greatly reduces the size of the sample file, but it can have a significant impact on sound quality.

The "About" menu selection brings up a window which explains that the Perfect Sound editor is copyrighted shareware, for which SunRize asks a donation of $20. A brief advertisement for the Perfect Sound package is also included. This notice is included for those people who have obtained the editor from a BBS or one of the commercial networks.

OTHER RESOURCES

Perfect Sound is very much a "hacker's package", and appears to be aimed squarely at people who want to add sampled sound to their own programs. The sound editor, as mentioned, includes several operations which are very useful for building composite sound effects. In addition to this, complete source code in C, and support routines in assembler, are provided for the editor. This is a useful addition for anyone interested in writing their own code, because they can use this material as a reference. And of course, if you don't like something about the sound editor you can change it, or add features you need.

SunRize has also included the file Format_IFF, which is the early 1985 specification document for IFF from Electronic Arts. This file is especially helpful for anyone who does not yet have the Exec manual, which includes a later IFF specification.

The disk also includes two dozen samples, most of which are simple demonstration sound effects.

ON THE OTHER HAND

Once again, it's time for me to list all the things I'd rather see done differently.

As far as the hardware is concerned, the lack of a feedthrough port is going to be an inconvenience for many. Lots of folks have parallel printers connected, and that means cable swapping when it's time to do a little sampling. Plus it eliminates the possibility of simply pushing the sampler window into the background and printing a file or two.

If you have your external drive on top of your computer to the right of the monitor, you'll find reaching the gain controls on the digitizer a bit awkward. And, because of the length of the box, it's fairly easy to wiggle the digitizer right out of the parallel port while trying to adjust the gain controls or change audio connections; best support it with one hand to be safe.

When changing the record sample rate, the system changes the playback sample rate accordingly. Problem is, it doesn't do a very good job of it. The default record period of 352 (just over 10 KHz) and default playback rate of 293 faithfully reproduces the recorded sample. Change the record rate to 127 and the playback rate changes to 105, which is way out in left field: the correct playback rate is closer to 200. If you intend to record at anything other than the default rate, you'll have to tweek the playback speed to get everything to come out right.

There doesn't appear to be any way to abort a playback. This can be inconvenient if you have a very long sample in memory, or have accidently set a playback rate which is far too slow: you have to wait out the complete playback before continuing.

Confirmation of certain operations would have been a nice touch. For example, selecting "Discard This Sample" will do just that... without consulting you. It's pretty easy to accidently select this choice when going for the ones above or below it if you are working quickly, or to mistakenly delete the wrong sample. (Or, at least it was for this "all thumbs" reviewer!) Still, using care and thinking ahead can avoid any problems. The program apparently doesn't check to see what font is being used, because booting with a stock WorkBench disk (using the 60 column font) does strange things to the drag bar and the two Play buttons. This is not terribly important, since most people use the 80 column font.

Perfect Sound is not very stable under 1.2. Digitizing a sample after startup and then immediately attempting to clip the front end from it invariably crashed my system. Thinking it might be interference from one of the background tasks I always have running, I tried booting from a stock 1.2 WorkBench disk. I got farther, but the program still crashed. On one occasion, the program itself reported "Bug... Code 02"; in all other cases I got a consistent Guru Meditation number which indicated problems with the Exec Library. Under 1.1 the program was very solid.

What was more fascinating to me was that I couldn't coax the program into sampling with 512K. I have an Insider 1 Meg RAM/RTC board from Michigan Software, but a supplied program allows me to toggle this extra RAM off, making it unavailable to the system. In this state the program would load and run, but would refuse to sample, insisting that it was "Out of Memory". Again, under 1.1 the program worked as it should, with or without the added memory. **Never, ever** swap disks while monitoring or recording a signal. The system gets confused, and the guru will insist your disk has been corrupted.

IN CLOSING

The Perfect Sound digitizer is a fairly nice package, and is a viable choice if you want to add sounds to your own programs. It's not very good at making instruments, and there is no convenient way to play back any instruments you make. Where this package excels, instead, is in it's ability to make special purpose sound effect and music samples. The fact that this is the least expensive digitizer of the bunch and is the only one which records in stereo right out of the box is squarely in its favor.

Even if you already have another sampler, it would be to your advantage to pick up a copy of the Perfect Sound editor from a network or bulletin board and send SunRize the $20 shareware payment; this will give you another flavor of editor with some capabilities the others don't have.

Nybbles,
Rick

•AC•

Summary...

This package works very well, although it is far from stable under 1.2. It is perhaps best suited to recording background music and sound effects for games and simulations, although it does have limited capabilities for making IFF instruments. The software portion of the package is available as shareware, and should be considered as a possible addition to your sampler tool kit.

Perfect Sound Digitizer $89.95
(Software only, $20 shareware)
COPY PROTECTION: None
REQUIREMENTS: Amiga with 512K, one drive, KS 1.1 (Marginal under 1.2).

SunRize Industries
PO Box 1453
College Station, TX 77841
409-846-1311

Basic Input

by Bryan Catley

Build a fully featured input routine which you may use in almost any situation, from almost any AmigaBasic program.

Traditionally, when a Basic program requires some input from the user, it uses one of the INPUT, LINE INPUT, or INPUT$ statements. While these statements work well and do what they are supposed to, they do have some serious disadvantages. When INPUT is used, the user MUST supply the exact number of required pieces of information, in the required sequence, in the correct format, and separated by commas. No leeway is given for errors. Should the user make an error (as they/we are prone to do), a rather cryptic error message is automatically generated which is unfriendly and which will destroy a neatly formatted screen. To avoid these problems, most people revert to LINE INPUT which accepts whatever characters the user enters. However, it now becomes necessary to edit the input and, if appropriate, to convert the input string to a numeric format.

An alternative is to use "string$=INPUT$(n)" where the user enters "n" number of characters and all entries, including control characters such as RETURN, BACKSPACE, etc., are returned in "string$" automatically. Now the program has to both edit the data and concern itself with handling the control characters correctly!

Besides all this, these three statements have one other serious disadvantage when used with the Amiga -- they lock out the mouse! A mouse click is remembered, but is not given to the program until the INPUT statement has completed! Try the following short program:

```
ON MOUSE GOSUB ClickRtn:MOUSE ON
   LINE INPUT "Enter something: ";x$
   PRINT "You entered: ";x$
   MOUSE OFF
   END
   ClickRtn:
   PRINT "Button Clicked!"
   RETURN
```

When you run this program, press the left mouse button after receiving the "Enter something:" prompt, but before you press the RETURN key. Note that nothing happens. Now press the RETURN key, and the "Button Clicked!" message will be displayed! The LINE INPUT effectively also becomes a MOUSE STOP statement, while pressing the RETURN key causes an effective MOUSE ON! This "feature" is particularly annoying if you have a requester routine where the user should have the option of entering some information or clicking on a gadget.

The "GetIp" Subprogram

All of these problems (except the trivial data conversion) are solved by the "GetIp" subprogram described in this article. "GetIp" will provide its users with the following capabilities:

- an initial data string may optionally be presented for editing
- inputed data is automatically edited for character, real (numeric with decimals), or integer (numeric without decimal), as requested
- left and right cursor keys are fully supported
- inputed data is inserted at the position of the cursor
- BACKSPACE will delete the character to the left of the cursor
- DEL will delete the character under the cursor
- ESC will clear the string to null
- automatic termination when the left mouse button is clicked; (provided the appropriate shared variable has been set by an external mouse event routine)

"GetIp" may be invoked in either of two ways:

CALL GetIp (Text$,datatype$,maxlen%)

or GetIp Text$,datatype$,maxlen%

where:

Text$ is a string variable containing a null value, or a string to be edited.

datatype$ is one of "CHAR", "REAL", or "INT" specified as a literal or as a string variable. This parameter determines the type of editing that will be performed. Any other value will effectively turn input editing off altogether, allowing such keys as the function keys to be used.

continued...

maxlen% is the maximum length that Text$ may assume, specified as an integer variable or literal.

Beside using the three given parameters, "GetIp" also assumes the user has:

- LOCATEd to the first position of the text area, whether or not any text is present
- set the shared variable "TxtCol" to the desired palette number for the text color
- set the shared variable "NewCur" to a non-zero value when it is desired to change the color of the cursor

Once set, the shared variables "TxtCol" and "NewCur" will, in most cases, not require being set again (which is why they are not parameters). However, it will occasionally be appropriate to change the cursor color, especially if the text or background colors change.

The cursor is originally drawn with the LINE statement, using the palette which is one less than the "TxtCol" palette (or one more if TxtCol=0). From then on it is drawn using PUT with the XOR (eXclusive OR) option. This method allows the character which is being covered by the cursor to "show through", and also allows the cursor to be erased simply by redrawing it in the same location. This provides all the requirements we need for editing but, because of the way XOR works, the actual colors which appear on the screen may be other than those expected. This is because the colors displayed depends on the original color specified for the cursor, and the colors of the character and background being drawn over! In addition to this, if the text or background color should end up the same color as that of the cursor, the cursor will become invisible!

Now, because of this, it is appropriate to have a method of redrawing the cursor even though it probably won't be necessary very often. As a general rule of thumb, if you change "TxtCol", you should probably also change the cursor color. As a matter of fact, "GetIp" could very easily be modified to do this automatically, but it is shown in the form which provides the most flexibility.

A First Example

Before we examine how "GetIp" works, let's see it in action with an example. First type in Listing #1 (the "GetIp" subprogram) and save it with the ",A" option. That is, when you have entered Listing #1, use an immediate command of the form SAVE "GetIp",A. This causes it to be saved in an ASCII format, which in turn, allows it to be MERGEd with other programs at a later time. (The menu save does not support the "A" option).

Note...

As shown, "GetIp" contains a lot of comments and blank lines which have been included for documentation and readability purposes. While you may, or may not, choose to enter all these extras, I would strongly recommend that you remove them from your final working version. Comments take up valuable memory space (yes, you should still be concerned about wasting memory), and slowdown execution. Since "GetIp" is involved with keyboard input, the latter is not too important, but the former certainly is.

Next, enter Listing #2 (the first example program), and then merge "GetIp" with it. You do this by clicking in the Basic Output window, and then entering the immediate command: MERGE "GetIp". "GetIp" is now read into memory and appended to the program already there. If you get an error message in response to the MERGE command you probably forgot to save it with the "A" option. No problem; just save Listing #2, load "GetIp" and re-save it with the "A" option, then load Listing #2, and repeat the MERGE command! Now save the combined programs before attempting to execute them; (you don't need the "A" option now).

Time to try it out! Select the RUN menu item, or enter the immediate command: RUN. Assuming you have no typos, you will receive a series of four prompts for input, and as each prompt is responded to, the program will print the data it actually receives, and go on to the next prompt. The first prompt is for character data; the second is also for character data, but an initial string is provided for editing; the third is for a real number; and the fourth is for an integer number.

Check out the features! Enter valid and invalid characters; try a function key when CHAR is requested; or an alphabetic when either REAL or INT is requested; or a period when INT is requested; etc. Cursor back and forth across the string; use the BACKSPACE key to delete the character to the left of the cursor and use the DEL key to delete the character under the cursor. Type a character to insert it at the cursor position; or press the ESC key to completely clear the string you are working with! Finally, note the various color combinations. Each example uses a different combination of the four Workbench colors as set in Preferences.

How "GetIp" Works

"GetIp" is really very straightforward, and I suspect you obtained a reasonable idea of how it worked when you entered it. However, the following may help to clear up any questions you may still have.

First of all, the three shared variables are set up. "TxtCol" and "NewCur" have already been discussed, but "MouseInd" is the variable that must be set to a non-zero value (by an external mouse event routine) in order for "GetIp" to terminate when the left mouse button is pressed. Following this, some of the variables which are "local" to "GetIp are initialized: "start" is set to the starting column number, "cur" (the relative position of the cursor over the text string) is set to zero, the foreground color is set to "TxtCol", and the pixel coordinates of the upper left-hand corner of the first character position are calculated (to be used to initially draw the cursor).

The very first time "GetIp" is used, "NewCur" is set to 1 to insure the cursor is initially drawn, and an array in which the cursor will be stored after it has been drawn is dimensioned. The size shown should be large enough to hold the cursor, regardless of the depth of the screen; (the higher the depth, the larger an array must be to hold the cursor). Next, and if a new cursor has been requested, it is drawn using the LINE statement, and saved in the previously dimensioned array with the GET statement. This will cause a momentary flicker of the cursor which you will probably not notice unless you are specifically looking for it.

We are now in the main processing part of the subprogram, and it starts by displaying the contents of the given text string, padded on the right with sufficient number of spaces to make the length of the string equal the given maximum length. Immediately following this, the cursor is PUT at the appropriate position using the XOR option. Once this has been done, we immediately LOCATE back to the start of the string, and calculate the pixel location of the cursor.

Now we wait for "MouseInd" to become non-zero, or for the user to press a key. When either occurs, a check is made to see exactly what happened. If "MouseInd" became non-zero, we go to the exit right away. However, if a key was pressed, we check to see if it was one of the control keys we are supporting. If it was, we branch to the appropriate routine; if not, we assume it to be a data character.

Data characters are then edited based on the data type provided by the user. Errors are indicated by a BEEP and a return is made to wait for another character. Valid characters are then inserted in the string provided the string is not yet at its maximum length. The insertion is effected by splitting the string immediately to the left of the cursor, and then by concatenating the left part, the character entered, and the right part. The relative cursor position is also incremented by one. A return is then made to display the new string with the cursor at its new position.

The remaining routines simply support the various editing functions. "CurRight" and "CurLeft" simply increment or decrement the relative cursor position by one respectively and then return to display the string. "DelLeft" (for the BACKSPACE key) splits the string one position to the left of and at the cursor, then rejoins the two pieces dropping the character that was to the immediate left of the cursor, and decrements the relative cursor position by one. "DelRight" (for the DEL key) splits the string on either side of the cursor and then rejoins the two pieces dropping the character which was under the cursor; the relative cursor position remains unchanged. (Note, in all of the above routines, if the cursor is at the first or last position of the string, a special case is presented). The "ClrTxt" routine simply prints spaces over the existing display and then blanks and zeros all the appropriate variables.

Just before "GetIp" is exited, the cursor is removed from the display by PUTting it on top of itself. (Another advantage of XOR is that if a shape is PUT, it appears; if it is PUT again on top of itself, it disappears! Note that this only works with the XOR mode though). This particular function is not always necessary, but is appropriate if the requester has several items to be input, and "GetIp" is used for each one individually. After all, it would look very messy to leave the cursor on item 1 while requesting item 2!

Now we have a better understanding of how "GetIp" works, let's look at another example.

A More Practical Example

Listing #3 is a short program which displays a requester in the upper left-hand corner of the screen. To enter or edit text in the text-box, first click in it (the cursor will appear), and then enter and/or edit the text as necessary. When done, you may press the RETURN key or click in the OK gadget to accept the text, or click in the Cancel gadget to abort the whole thing. Note that you have the option of either pressing the RETURN key or clicking the OK gadget to accept the text.

Enter Listing #3 and MERGE "GetIp" with it just as you did with Listing #2. Remember to save a copy before you try running the program.

Aside...

Listing #3 makes use of the three gadget subprograms (BldGadgets, DrawGadgets, and GetGadget) which I described in another article (Basic Gadgets) in Amazing Computing. Should you already have entered these subprograms, there is no need to do it again. Just MERGE in the ones you have! If you don't already have them then you'll need to type them in; and if you're interested in finding out more about them, then you'll need to refer to the other article.

The "DoRequester" subprogram is pretty simple but it might very well form the basis for your own, somewhat more flexible, requester subprogram. The only things worth commenting on are the list of SHARED variables (it shares some with the gadget subprograms and some with "GetIp"), and the fact that the mouse event subroutine is in the main program, even though it is only required in this subprogram. It appears that event routines are not supported in subprograms, (although I have not done an exhaustive test on this), and must be placed in the main program. Anyway, you now have a fully featured input routine which you may use in almost any situation, from almost any AmigaBasic program.

Some Final Comments

If you have been following along for the last few months you have started to build a collection of subprograms which provide the basis for making truly professional looking programs. You have subprograms to display a title screen giving your programs a distinctive look, to use gadgets exactly as desired, and now you have an input routine which does everything you need and a simple requester subprogram. I think you'll find yourself using these subprograms in almost every program you write!

continued...

If you are reading *Amazing Computing™* for the first time, you have not seen *Amazing Computing™*. Look what you have missed!

Volume 1 Number 1 Premiere February 1986

Super Spheres By Kelly Kauffman An ABasic Graphics program
Date Virus By John Foust There is a disease that may attack your Amiga
EZ-Term by Kelly Kauffman An ABasic Terminal program
Miga Mania by Perry Kivolowitz Programming fixes and mouse care
Inside CLI by George Musser a guided insight into the AmigaDos™
CLI Summary by George Musser Jr. A removable list of CLI commands
AmigaForum by Bela Lubkin A quick trip through Compuserve's Amiga SIG
Commodore Amiga Development Program by Don Hicks
Amiga Products A listing of present and expected products.

Volume 1 Number 2 March 1986

Electronic Arts Comes Through A look at the new software from EA
Inside CLI: part two by George Musser George continues his investigation of CLI and ED
A Summary of ED Commands
Live! by Rich Miner A review of the Beta version of the Live! frame grabber
Online and the CTS Fabite 2424 ADH Modem reviewed by John Foust
Superterm V 1.0 By Kelly Kauffman A terminal program written in Amiga Basic
A Workbench "More" Program by Rick Wirch
Amiga BBS numbers

Volume 1 Number 3 April 1986

Analyze! a review by Ernest Viveros
Reviews of Racter, Barataccas and Mindshadow
Forth! The first of our on going tutorial
Deluxe Draw!! by Rich Wirch An Amiga Basic program for the artist in us all.
Amiga Basic, A beginners tutorial
Inside CLI: part 3 by George Musser George gives us PIPE

Volume 1 Number 4 May 1986

SkyFox and Articfox Reviewed
Build your own 5 1/4 Drive Connector By Ernest Viveiros
Amiga Basic Tips by Rich Wirch
Scrimper Part One by Perry Kivolowitz A C program to print your Amiga screen
Microsoft CD ROM Conference by Jim O'Keane
Amiga BBS Numbers

Volume 1 Number 5 1986

The HSI to RGB Conversion Tool by Steve Pietrowicz Color manipulation in BASIC
AmigaNotes by Rick Rae The first of the Amiga music columns
Sidecar A First Look by John Foust A first "under the hood" look
John Foust Talks with R. J. Mical at COMDEX™
How does Sidecar affect the Transformer an interview with Douglas Wyman of Simile
The Commodore Layoffs by John Foust John looks at the "cuts" at Commodore
Scrimper Part Two by Perry Kivolowitz
Marauder reviewed by Rick Wirch
Building Tools by Daniel Kary

Volume 1 Number 6 1986

Temple of Apshai Triology reviewd by Stephen Pietrowicz
The Halley Project: A Mission in our Solar System reviewed by Stephen Pietrowicz
Flow: reviewed by Erv Bobo
Textcraft Plus a First Look by Joe Lowery
How to start your own Amiga User Group by William Simpson
Amiga User Groups
Mailing List by Kelly Kauffman a basic mail list program
Pointer Image Editor by Stephen Pietrowicz
Scrimper: part three by Perry Kivolowitz
Fun With the Amiga Disk Controller by Thom Sterling
Optimize Your AmigaBasic Programs for Speed by Stephen Pietrowicz

Volume 1 Number 7 1986

Aegis Draw: CAD comes to the Amiga by Kelly Adams
Try 3D by Jim Meadows an introduction to 3D graphics
Aegis Images/ Animator: a review by Erv Bobo
Deluxe Video Construction Set reviewed by Joe Lowery
Window requesters in Amiga Basic by Steve Michel
ROT by Colin French a 3D graphics editor
"I C What I Think" Ron Peterson with a few C graphic programs
Your Menu Sir! by Bryan D. Catley programming menues in Amiga Basic
IFF Brush to AmigaBasic 'BOB' Basic editor by Michael Swinger
Linking C Programs with Assembler Routines on the Amiga by Gerald Hull

Volume 1 Number 8 1986

The University Amiga By Geoff Gamble Amiga's inroads at Washington State University
MicroEd a look at a one man army for the Amiga
MicroEd, The Lewis and Clark Expedition reviewed by Robert Frizelle
Scribble Version 2.0 a review
Computers in the Classroom by Robert Frizelle
Two for Study by Robert Frizelle a review of Discovery and The Talking Coloring Book
True Basic reviewed by Brad Grier
Using your printer with the Amiga
Marble Madness reviewed by Stephen Pietrowicz
Using Fonts from AmigaBasic by Tim Jones
Screen SaVer by Perry Kivolowitz A monitor protection program in C
Lattice MAKE Utility reviewed by Scott P. Evernden
A Tale of Three EMACS by Steve Poling
.bmap File Reader in Amiga Basic by Tim Jones A look into the .bmap files

Volume 1 Number 9 1986

Instant Music Reviewed by Steve Pietrowicz
Mindwalker Reviewed by Richard Knepper
The Alegra Memory Board Reviewed by Rich Wirch
TxEd Reviewed by Jan and Cliff Kent
Amazing Directory A guide to the sources and resources
Amiga Developers A listing of Suppliers and Developers
Public Domain Catalog A condensed listing of Amicus and Fred Fish PDS Disks
Dos 2 Dos review by Richard Knepper Transfer files from PC/MS-DOS and AmigaBasic
MaxiPlan review by Richard Knepper The Amiga version of Lotus 1-2-3
Gizmoz by reviewed by Peter Wayner A collection of Amiga extras!
The Loan Information Program by Brian Catley basic prog. to for your financial options
Starting Your Own Amiga Related Business by William Simpson
Keep Track of Your Business Usage for Taxes by James Kummer
The Absoft Amiga Fortran Compiler reviewed by Richard A. Reale
Using Fonts from AmigaBasic, Part Two by Tim Jones
68000 Macros on the Amiga by Gerald Hull Advance your program's ability.
TDI Modla-2 Amiga Compiler by Steve Faiwiszewski Looking at an alternative to C

Volume 2 Number 1 1987

What Digi-View Is... Or, What Genlock Should Be! by John Foust
AmigaBasic Default Colors by Bryan Catley
AmigaBasic Titles by Bryan Catley
A Public Domain Modula-2 System reviewed by Warren Block
One Drive Compile by Douglas Lovell using Lattice C with a single drive system
A Megabyte Without Megabucks by Chris Irving An Internal Megabyte upgrade
Digi-View reviewed by Ed Jakober
Defender of the Crown reviewed by Keith Conforti
Leader Board reviewed by Chuck Raudonis
Roundhill Computer System's PANEL reviewed by Ray Lance
Digi-Paint....by New Tek previewed by John Foust
Deluxe Paint II ....from Electronic Arts previewed by John Foust

Volume 2 Number 2 1987

The Modem by Josph L. Rothman efforts of a BBS Sysop
MacroModem reviewed by Stephen R. Pietrowicz
GEMINI or "It takes two to Tango" by Jim Meadows Gameing between machines
BBS-PC! reviewed by Stephen R. Pietrowicz
The Trouble with Xmodem by Joseph L. Rothman
The ACO Project....Graphic Teleconferencing on the Amiga by S. R. Pietrowicz
Flight Simulator II....A Cros Country Tutorial by John Rafferty
A Disk Librarian in AmigaBASIC by John Kennan
Creating and Using Amiga Workbench Icons by Celeste Hansel
AmigaDOS version 1.2 by Clifford Kent
Amiganotes...The Amazing MIDI Interface build your own by Richard Rae
AmigaDOS Operating System Calls and Disk File Management by D. Haynie
Working with the Workbench by Louis A. Mamakos Programming in C

Volume 2 Number 3

The Amiga 2000™ by John Foust A First look at the new high end of the Amiga™ line.
The Amiga 500™ by John Foust A look at the new low priced Amiga
An Analysis of the New Amiga PCs by John Foust Speculation on the New Amigas
Gemini Part II by Jim Meadows The concluding article on two-player games
Subscripts and Superscripts in AmigaBASIC by Ivan C. Smith
The Winter Consumer Electronics Show by John Foust
AmigaTrix by Warren Block Those little shortcuts that make using the Amiga™ easier
Intuition Gadgets by Harriet Maybeck Tolly A journey through gadget-land using C
Shanghai reviewed by Keith M. Conforti
Chessmaster 2000 & Chessmate reviewedby Edwin V. Apel, Jr.
Zing! from Meridian Software reviewed by Ed Bercovitz
Forth! by Jon Bryan Get stereo sound into your Forth programs.
Assembly Language on the Amiga™ by Chris Martin
Roomers by theBandito Genlocks are finally shipping, and MORE!!!
AmigaNotes by Richard Rae Hum Busters... "No stereo? Y not?..."
The AMICUS Network by John Foust "CES, user group issues and Amiga Expo"

Volume 2 Number 4 1987

Amazing Interviews Jim Sachs by Steve Hull Amiga Artist extrodinaire
The Mouse That Got Restored by Jerry Hull and Bob Rhode Mouse repair
Sluething Public Domain Disks with CLI by John Foust using CLI on PDS
Highlights from the San Francisco Commodore Show by Steve Hull
Speaker Sessions at the San Francisco Commodore Show by Harriet Tolly
The Household Inventory System in AmigaBASIC™ by Bryan Catley
Secrets of Screen Dumps by Natkun Okun
Using Function Keys with MicroEmacs by Greg Douglas
Amigatrix II by Warren Block More shortcuts to make the Amiga work easier
Basic Gadgets by Brian Catley Create your own gadget functions
Gridiron reviewed by Keith Conforti Real Tackle football for the Amiga
Star Fleet I Version 2.1 reviewed by John Tracy Space in the Amiga
The TIC reviewed by John Foust A tiny battery powered Clock Calendar
Metascope reviewed by Harriet Tolly An easy-to-use debugger by Metadigm

To Be Continued.....

Amazing Computing™

Your Resource to the Commodore Amiga™

Amazing Computing™ has been offering the Amiga community the best in technical knowledge and reviews for the Commodore-Amiga™ since our first issue in Febuary 1986.

We were the first magazine to document CLI
We were the first to show Sidecar™ from COMDEX™ in full detail.
We were the first to document a 5 1/4 drive connector
We were the first with a 1 Meg Amiga upgrade hardware project!
We were the first magazine to offer serious programming examples and help.
We were the first magazine to offer Public Domain Software at reasonable prices.
We were the first magazine with the user in mind!

However, Amazing Computing™ will not rest on past achievements. The Commodore-Amiga™ has more surprises for you and we are ready to cover them. We even have a few tricks that will "Amaze" you.

To subscribe to Amazing Computing™ or to purchase Public Domain Software, please fill out the form below and send with a Check or money order to:

PiM Publications Inc.
P.O.Box 869
Fall River, MA 02722.

Back issues are still available at $4.00 each (Foreign orders, please add $1.00 U.S. per issue for P. & H). All payments must be made by check or money order in U.S. funds drawn on a U.S. Bank.

Yes! *Amaze* Me!

Please start my subscription to Amazing Computing™ with the next available issue. I have enclosed $24.00 for 12 issues in the U.S. ($30.00 Canada and Mexico, $35.00 overseas). All funds must be in U.S. Currency drawn on a U.S. Bank.

Please circle your selection:
Subscription PDS (as noted) Back Issues Sub Renewal

Name _____
Street _____
City _____ **St.** ____ **ZIP** _____
Amount enclosed _____

Mass. Residents, please add 5% sales tax on PDS orders

AC587

Public Domain Software:
$6.00 each for subscribers (yes, even new ones!)
$7.00 each for non subscribers.

AMICUS: A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17
Fred Fish: FF1 FF2 FF3 FF4 FF5 FF6 FF7 FF8 FF9 FF10 FF11 FF12 FF13 FF14 FF15 FF16 FF17 FF18 FF19 FF20 FF21 FF22 FF23 FF24 FF25 FF26 FF27 FF28 FF29 FF30 FF31 FF32 FF33 FF34 FF35 FF36 FF37 FF38 FF39 FF40 FF41 FF42 FF43 FF44 FF45 FF46 FF47 FF48 FF49 FF50 FF51 FF52 FF53 FF54 FF55 FF56 FF57 FF58

BACK ISSUES: $4.00 each (foreign orders add $1.00 each for Postage and Handeling
VOL.1#1 VOL.1#2 VOL.1#3 VOL.1#4 VOL.1#5 VOL.1#6 VOL.1#7 VOL.1#8 VOL.1#9 VOL.2#1 VOL.2#2 VOL.2#3 VOL.2#4

Now remember, these subprograms are not cast in concrete. They don't have to be used exactly as is in every program. In many cases it will make a lot of sense to make some changes to customize them to the specific needs of the programs using them; after they have been MERGEd with them of course! For example, with "GetIp" it may be desirable to force the cursor to be drawn in a specific color, in order to obtain the best color combinations between text and cursor.

Listing One

```
' Listing #1 -- "GetIp"
'
' "GetIp" is a keyboard input routine that works in
conjunction with
' the mouse.  The using program must set the shared variable

' "MouseInd" to a non-zero value (via an external mouse
event
' routine) for "GetIp" is terminate automatically.
'
' The three parameters are:
'    -a string containing the text to edited or null
'    -a data type indicator; "CHAR", "REAL", or "INT"
'    -the maximum length of the string as an integer
'
' "GetIp" also expects the user:
'    -to have LOCATEd to start of text area; even if text
exists
'    -to have set "TxtCol" to PALETTE number for text color
'    -to have set "NewCur" to a non-zero value if it is
desired
'     to redraw the cursor
'
' Bryan D. Catley
' October 1986
'
SUB GetIp (Text$,DataType$,maxlen%) STATIC
SHARED TxtCol,NewCur,MouseInd
start=POS(0):cur=0:COLOR TxtCol
xpix=(start-1)*8:ypix=(CSRLIN-1)*8
IF FirstTime=0 THEN FirstTime=1:NewCur=1:DIM IPcursor%(46)
IF NewCur=1 THEN
  NewCur=0
  CurCol=TxtCol-1:IF CurCol<0 THEN CurCol=TxtCol+1
  LINE(xpix,ypix)-STEP(7,7),CurCol,bf
  GET(xpix,ypix)-STEP(7,7),IPcursor%
END IF
ShoText:
GOSUB DisplayText
NxtChar:
x$="":MouseInd=0:LeftPart$="":RightPart$=""
WHILE x$="" AND MouseInd=0:x$=INKEY$:WEND
IF MouseInd<>0  THEN GetDone  ' Mouse was clicked
IF x$=CHR$(30)  THEN CurRight ' Right-cursor
IF x$=CHR$(31)  THEN CurLeft  ' Left-cursor
IF x$=CHR$(8)   THEN DelLeft  ' Back-space key
IF x$=CHR$(127) THEN DelRight ' Delete key
IF x$=CHR$(27)  THEN ClrText  ' Escape key
IF x$=CHR$(13)  THEN GetDone  ' Return key

' Validate character entered
IF DataType$="CHAR" THEN
  IF x$<CHR$(32) OR x$>CHR$(127) THEN
    BEEP:GOTO NxtChar
  END IF
ELSEIF DataType$="REAL" THEN
  IF (x$<CHR$(48) OR x$>CHR$(57)) AND (x$<>".") THEN
    BEEP:GOTO NxtChar
  END IF
ELSEIF DataType$="INT" THEN
  IF x$<CHR$(48) OR x$>CHR$(57) THEN
    BEEP:GOTO NxtChar
  END IF
END IF

' We must insert character at cursor position
InsertChar:
IF LEN(Text$)=maxlen% THEN BEEP:GOTO NxtChar
IF cur>0 THEN LeftPart$=MID$(Text$,1,cur)
IF LEN(Text$)>0 THEN RightPart$=MID$(Text$,cur+1,LEN(Text$)-
LEN(LeftPart$))
Text$=LeftPart$+x$+RightPart$:cur=cur+1
GOTO ShoText

' Move cursor right one position
CurRight:
IF cur=LEN(Text$) THEN NxtChar
cur=cur+1:GOTO ShoText

' Move cursor left one position
CurLeft:
IF cur=0 THEN NxtChar
cur=cur-1:GOTO ShoText

' Delete one character to left of cursor; (BACK SPACE)
DelLeft:
IF LEN(Text$)=0 OR cur=0 THEN BEEP:GOTO NxtChar
IF cur>1 THEN LeftPart$=MID$(Text$,1,cur-1)
IF LEN(Text$)>cur THEN
RightPart$=MID$(Text$,cur+1,LEN(Text$)-cur)
Text$=LeftPart$+RightPart$
cur=cur-1:GOTO ShoText

' Delete character cursor is covering; (DELETE)
DelRight:
IF LEN(Text$)=0 OR cur=LEN(Text$) THEN BEEP:GOTO NxtChar
IF cur>0 THEN LeftPart$=MID$(Text$,1,cur)
IF cur+1<LEN(Text$) THEN
RightPart$=MID$(Text$,cur+2,LEN(Text$)-cur+1)
Text$=LeftPart$+RightPart$
GOTO ShoText

' Clear text and start over
ClrText:
PRINT SPACE$(maxlen%+1);:LOCATE ,start
cur=0:Text$="":GOTO ShoText

' Display text as it exists
DisplayText:
PRINT Text$+SPACE$(maxlen%+1-LEN(Text$));:LOCATE ,start
xpix=(start+cur-1)*8:PUT(xpix,ypix),IPcursor%
RETURN

' Return was pressed
GetDone:
PUT(xpix,ypix),IPcursor%
END SUB
```

Listing Two

```
' Listing #2 -- "GetIp" Demo
' Bryan D. Catley
' October 1986
'
MouseInd=0:NewCur=0:PRINT"Start GetIp Demo"

Wot$="":LOCATE 3,5:PRINT"CHAR:";
LOCATE ,10:TxtCol=3:GetIp Wot$,"CHAR",10
COLOR 1,0:LOCATE 4,1:PRINT"Received=";Wot$

Wot$="ABCD":LOCATE 6,5:PRINT"CHAR:";
LOCATE ,10:TxtCol=3:COLOR TxtCol,1
PRINT SPACE$(16);:LOCATE ,10
```

```
GetIp Wot$,"CHAR",15
COLOR 1,0:LOCATE 7,1:PRINT"Received=";Wot$

Wot$="":LOCATE 9,5:PRINT"REAL:";
LOCATE ,10:TxtCol=2:COLOR TxtCol,3:NewCur=1
PRINT SPACE$(11);:LOCATE ,10
GetIp Wot$,"REAL",10:n=VAL(Wot$)
COLOR 1,0:LOCATE 10,1:PRINT"Received=";n

Wot$="":LOCATE 12,5:PRINT" INT:";
LOCATE ,10:TxtCol=0:COLOR TxtCol,2:NewCur=1
PRINT SPACE$(6);:LOCATE ,10
GetIp Wot$,"INT",5:n=VAL(Wot$)
COLOR 1,0:LOCATE 13,1:PRINT"Received=";n

LOCATE 15,1:PRINT"End GetIp Demo"
END
```

Listing Three

```
' Listing #3
'
' Demonstration of USING the "Gadget" AND "GetIp" subprograms

' in a simple program requester situation.
' Bryan D. Catley
' October 1986
'
NumGdgts=3:ReqRslt%=0:TxtString$=""
DIM bx(NumGdgts,7),bxtxt$(NumGdgts)
BldGadgets NumGdgts,bx(),bxtxt$()
DATA  14, 31,176, 9,1,0,-2,""
DATA  20, 52, 24,16,1,0, 2,"Ok"
DATA 124, 52, 56,16,1,0, 2,"Cancel"
'
' Program does its thing until it needs a requester

TxtString$="Unimaginative Name"
'TxtString$=""
Prompt1$="Enter desired value..."
Prompt2$="then CR or click OK"
DoRequester TxtString$,Prompt1$,Prompt2$,ReqRslt%
ON ReqRslt% GOTO DoOk,DoCancel

'Handle results of Requester for our demo
DoOk:
PRINT"You pressed RETURN, or clicked OK"
PRINT"String received was: ";TxtString$
GOTO Quit
DoCancel:
PRINT"You clicked CANCEL."
GOTO Quit

' Mouse Event routine
ReqMouse:
GetGadget 0,2,bx(),bxtxt$(),gdgt
RETURN

' Remainder of program body and program termination
'
Quit:
END

' Simple Requester Subprogram
'
SUB DoRequester (WkLine$,Prompt1$,Prompt2$,Result%) STATIC
SHARED bx(),bxtxt$(),gdgt,TxtCol,NewCur,MouseInd
WINDOW 9,"Program Request:",(0,0)-(200,80),16,-1
TxtCol=2:COLOR TxtCol,3:CLS
CharData$="CHAR":CharLen%=20
LOCATE 2,3:PRINT Prompt1$
LOCATE 3,3:PRINT Prompt2$
DrawGadgets 0,2,bx(),bxtxt$()
LOCATE 5,3:COLOR TxtCol,1:PRINT WkLine$
ON MOUSE GOSUB ReqMouse:MOUSE ON
MouseInd=0:NewCur=0:gdgt=0:WHILE gdgt=0:SLEEP:WEND
ON gdgt GOTO DoInput,DidOK,DidCancel
DoInput:
LOCATE 5,3:CALL GetIp (WkLine$,CharData$,CharLen%)
IF MouseInd=0 THEN DidCR
ON gdgt GOTO DoInput,DidOK,DidCancel
DidCR:
DidOK:
Result%=1:GOTO DRExit
DidCancel:
Result%=2:GOTO DRExit

DRExit:
MOUSE OFF:WINDOW CLOSE 9
END SUB

' Three Subprograms to provide easy use of Gadgets
' in Amiga Basic
'
SUB BldGadgets (Num,T1(),T2$()) STATIC
FOR n=0 TO Num-1
  FOR m=0 TO 6
    READ T1(n,m)
  NEXT m
  READ T2$(n)
NEXT n
END SUB

SUB DrawGadgets (Ga%,Gb%,T1(),T2$()) STATIC
FOR n=Ga% TO Gb%
  x1=T1(n,0):y1=T1(n,1):x2=x1+T1(n,2):y2=y1+T1(n,3)
  bg=T1(n,4):fg=T1(n,5):bo=T1(n,6)
  LINE(x1,y1)-(x2,y2),bg,bf:LINE(x1,y1)-(x2,y2),fg,b
  IF bo>-1 THEN
    LINE(x1+2,y1+2)-(x2-2,y2-2),fg,b
    LINE(x2+1,y1+1)-(x2+1,y2+1),bo
    LINE(x2+1,y2+1)-(x1+1,y2+1),bo
    COLOR fg,bg:row%=INT(y1/8+2):col%=INT(x1/8+2)
    LOCATE row%,col%:PRINT T2$(n)
  END IF
NEXT n
END SUB

SUB GetGadget (Ga%,Gb%,T1(),T2$(),type) STATIC
SHARED MouseX%,MouseY%,MouseInd
WHILE MOUSE(0)=0:WEND
r%=CSRLIN:c%=POS(0)
mx=MOUSE(1):my=MOUSE(2)
MouseX%=mx:MouseY%=my:MouseInd=0
FOR n=Ga% TO Gb%
  IF mx>T1(n,0) AND mx<T1(n,0)+T1(n,2) THEN
    IF my>T1(n,1) AND my<T1(n,1)+T1(n,3) THEN
      bg=T1(n,4):fg=T1(n,5):bo=T1(n,6)
      IF bo>-1 THEN
        x1=T1(n,0)+2:y1=T1(n,1)+2
        x2=x1+T1(n,2)-4:y2=y1+T1(n,3)-4
        LINE(x1,y1)-(x2,y2),fg,bf
        COLOR bg,fg:row%=INT(y1/8+2):col%=INT(x1/8+2)
        LOCATE row%,col%:PRINT T2$(n)
      ELSE
        IF bo=-1 THEN
          x1=T1(n,0):y1=T1(n,1):x2=x1+T1(n,2):y2=y1+T1(n,3)
         LINE(x1,y1)-(x2,y2),fg,bf:LINE(x1,y1)-(x2,y2),bg,b
        END IF
      END IF
      type=n-Ga%+1:n=Gb%:MouseInd=1
      IF bo>-1 THEN n%=type+Ga%-1
    END IF
  END IF
NEXT n
WHILE MOUSE(0)<>0:WEND
IF type<>0 AND bo>-1 THEN DrawGadgets n%,n%,T1(),T2$()
LOCATE r%,c%

END SUB
```

•AC•

Roomers

by the Bandito

Inside developer information says that...
According to one insider...
There are more rumors of...

The Amiga 500 and 2000 continue to generate controversy and rumors. At press time, a Commodore representative was in Hong Kong, overseeing the first production run of the Amiga 2000. No word on whether the 500 has begun production, so this rumor might bode ill against early appearance of the 500. Some say the high cost of the Sidecar - about $900 - is due to the higher cost of manufacturing in Germany. There are more rumors of FCC troubles, and others are blaming software delays for the eternal lateness of the machine.

The Amiga 500 may accept Amiga 1000 peripherals by using a small adapter that provides extra power and an extension to move the bus up and out away from the bottom of the 500 case. A similar set of extension cables could link the Amiga 500 to today's Amiga 1000 version of the Genlock. Another source said that the prospect is small for recognizing a Kickstart on power-up, and loading any other version of AmigaDOS into RAM space on the 500. The ROMs inside today's production units will not have this ability; they are straight AmigaDOS 1.2.

Inside developer information says that the Amiga 2000 does not run some off the shelf Amiga 1000 software. The early German production units use a slightly different keyboard controller. For programs and programmers that break all the rules of Amiga system programming and do not use the official AmigaDOS operating system calls, this spells trouble.

The Fat Agnes graphics chip used in the 500 and 2000 has a few extra pins, and some speculators think this might mean a direct-chip replacement upgrade to the higher resolution graphics chips in a future Amiga.

Other forward-looking Amiga owners want to upgrade their Amiga 1000, so it can have the same flashy expansion options as the Amiga 2000. Several manufacturers are planning expansion boxes for the Amiga 1000 that give both Amiga 2000-style Zorro slots and the basic PC compatible bus and slots. Such expansion works towards effectively bringing the 1000 in line with the 2000. It will mean another box on the side of your Amiga 1000, and you probably will not be allowed to place any peripherals between this box and the Amiga. The Commodore Bridge card will drop into these boxes, bringing PC compatibility. However, these boxes will not allow the proposed 68020 upgrade card.

Modem junkies have been getting big laughs from the Laurel and Hardy antics of Amiga developers on Compuserve. The mudslinging over hard disk and expansion bus specifications left a lot of dirty faces, and left many Amiga owners perplexed about what it all really meant.

Commodore is also working on a version of AmigaDOS that boots directly from hard disk. According to one insider, it takes less than fifteen seconds from power-on to hard disk activity. This version of AmigaDOS needs special V1.21 ROM chips.

The fast file system AmigaDOS customized for hard disks is on its way, too. Copies termed 'pre-alpha' are in the hands of hard disk manufacturers. According to one source who is using the new file system, Metacomco's Tim King is still working on it, tweaking hard disk performance by a minimum of three to four times, up to a maximum of twenty times. Current versions of the software have only optimized read access of the drives, and improved disk writes are in the works, but are not expected to gain much in performance, as compared to read access.

Modem prices are dropping, and according to one source, 2400 baud modems should be available for less than $200. Prices for internal PC compatible 2400 baud modems have dropped to this range, so external modems are not far behind.

•AC•

MORE...

INTUITION GADGETS

Boolean Gadgets

By Harriet Maybeck Tolly

Welcome back to the second installment of 'Intuition Gadgets'.

A quick refresher: Gadgets provide a user interface for your application. There are three types of Gadgets available:

1. *String Gadgets* (covered last month), which accept strings of ASCII text.
   a. *Integer Gadgets*, a type of *String Gadget* that accepts only integers.

2. *Boolean Gadgets*, hit and toggle types. Boolean gadgets are the topic of this article.

3. *Proportional Gadgets*, or sliders.

My intent with this series is not to regurgitate the manuals. So, if you are serious about using Gadgets, you will want the Intuition manual or a reasonable substitute. Unless you program using Intuition every day, it would help to review the structures associated with Gadgets.

I will be covering areas which can be confusing, have changed in V1.2, or that still contain bugs. As I stated last month, this is only a small part of Intuition. Gadgets in general are easy to program. The material I present to you here represents areas that may not be as straightforward as you would like.

So, let us look at Boolean Gadgets.

Boolean Gadgets are familiar to anyone who has clicked the OK or CANCEL box on a requester. Boolean Gadgets supply an on/off interface to the user. A 'hit' type Boolean Gadget is selected as long as the user holds the select (left mouse) button down over it. It goes back to the unselected state as soon as the button is released. A 'toggle' Boolean Gadget toggles between the on and off state. It remains selected or unselected until the user clicks on it again. The imagery associated with Boolean Gadgets can range from a simple box, to now, in V1.2, a complicated masked shape.

MUTUAL EXCLUSION

One of the most frequently heard complaints concerning Boolean Gadgets is the lack of implementation of Mutual Exclusion. The desired effect would be that if two Gadgets were Mutually Exclusive, clicking one on would automatically turn the other off and vice versa. For example, the male and female Gadgets in the famous SpeechToy demo simulate Mutual Exclusion very well. The voice is either male or female. It cannot be both. It cannot be neither.

It is true that this omission is frustrating, since Intuition seems to bend over backwards to provide mechinisms to do just about everything else. It is, however, very simple to implement on your own.

The Enhancer manual gives the official method for doing this. It asks that you follow some strict guidelines, and for good reasons, as we shall see later.

Gadgets can have various Activation Flags set. Among them are the following:

GADGIMMEDIATE
If this flag is set, you will get GADGETDOWN messages as soon as the user clicks the select (left mouse) button.

RELVERIFY
If this flag is set, you will get GADGETUP messages when the select button is released, provided it is still over the Gadget.

TOGGLESELECT
If this flag is set, Intuition toggles the selected state of the Gadget each time the user clicks on it.

When implementing Mutual Exclude, the Gadgets involved should have Activation set to GADGIMMEDIATE, and NOT TOGGLESELECT nor RELVERIFY.

Many of you may have seen examples of Mutual Exclusion using TOGGLESELECT Gadgets, a la SpeechToy. I have not had a problem with this, but Jim Mackraz, author of the V1.2 changes, gives a good argument for avoiding it and only using GADGIMMEDIATE Gadgets. If you use TOGGLESELECT Gadgets, you will be manipulating some of

continued...

the Flags, while Intuition is manipulating others. Since this is a 'fix' to begin with, relinquishing partial control is not the best approach. If both Gadgets are GADGIMMEDIATE, you will have complete responsibility for all manipulation of Flags, and know their states at all times.

Your program should choose one of the Gadgets to start out selected, by setting the SELECTED Flag of that Gadget. Reading the Intuition manual might give you the impression that the SELECTED Flag is meaningless in the case of a non-TOGGLESELECT Gadget. I have investigated this further, and my sources tell me that it is valid to manipulate the SELECTED Flag, even though we are using GADGIMMEDIATE (non-TOGGLESELECT) Gadgets.

When implementing Mutual Exclusion, the GadgetRender member of the Gadget structure should point to an Image; Pointing to a Border or to NULL will create problems. These problems are due to the simple refreshing method used by Intuition when an Image is absent from the Gadget. This is explained further in the next section.

Now, listen for IDCMP messages of type GADGETDOWN on the Gadgets. When the user clicks on the unselected Gadget, you will need to remove both Gadgets, set the SELECTED flag of the one they clicked, and clear the SELECTED Flag of the other Gadget. Conversely, if the user clicks on the selected Gadget, you will need to clear the SELECTED Flag of the Gadget the user clicked, and set the SELECTED Flag of the other Gadget. Now, add the Gadgets back in, and refresh them.

This can be done in the following way:

1. Remove the Gadgets to be selected/un-selected.
2. Set or Clear the SELECTED Flag. In C, this would look like:

```
BoolGadget_one.Flags ^= SELECTED;
BoolGadget_two.Flags ^= SELECTED;
```

This exclusive-OR will toggle the SELECTED states of the Gadgets.

3. Add the Gadgets back into the list.
4. Refresh the Gadgets' images on the Window.

The example program below contains a pair of Mutually Exclusive Gadgets.

REFRESHING NON-IMAGE GADGETS

As mentioned above, Mutual Exclude manipulation should only be performed on Gadgets with Images, highlighted by GADGHCOMP. This is a subset of an even bigger warning. Any type of manipulation of the SELECTED Flag, followed by calls to RefreshGadgets(), is tricky at best with non-Image Gadgets. If you try to manipulate the SELECTED Flag of a Gadget with a Border (or nothing), you will become quickly frustrated, as the colors of your Gadgets will appear to have a mind of their own. This is due to the VERY simple way that Intuition handles the complementing of Gadgets without Images.

For a Gadget without an Image, when you ask Intuition to do a RefreshGadgets(), it checks the SELECTED flag. If it is set, it determines that it must highlight the Gadget for you. It does this by complementing all the bits in the Gadget's select box. Now, if the Gadget was already in a SELECTED (highlighted) state when RefreshGadgets() was called, there is a problem. The bits get complemented back to the UN-highlighted state! Conversly, if you do a RefreshGadgets on a Gadget whose SELECTED Flag has been cleared, it will do nothing. In our above example of Mutual Exclude, assume we take the selected Gadget, clear its SELECTED Flag and Refresh it. This will cause no change in the imagery of the Gadget. To me this is a bug; To Commodore it is a matter of documentation.

The problem is magnified even further when your Gadget has GadgetText associated with it. Depending upon the drawmode, some parts of the text also get incorrectly complemented. A program in the public domain called MxGads, on Fred Fish #52, tries to address this problem. As I stated above, the problem is that complementing is done when you might not want it, and is not done when you do want it. MxGads solves the problem by calling RefreshGadgets() when it (seemingly) is not needed. Remember, if we call RefreshGadgets() first, while the Flag was still set, Intuition will complement it for us back to the UN-highlighted state.

So now when we clear the Flag and call RefreshGadgets, even though this second call DOES NOT complement the bits, it has already been done by the first call! This seems to be the developers' work-around of choice, as I have also seen this suggested on BIX. I'm still not sure I'd want to support a product with this kind of logic. I think a better solution is to follow the Enhancer manual's advice, and only play with this flag on Gadgets with Images.

For those of you not trying to fiddle with the SELECTED Flag, and getting flakey results simply from calling RefreshGadgets() (this is how I stumbled over it), be sure to use RefreshGList(). This new V1.2 function allows you to specify a particular range of Gadgets to refresh. This will help you avoid refreshing a questionable Gadget by accident. If, however, you really do want to refresh a SELECTED (highlighted) Boolean Gadget with a Border, know what you are getting into.

GADGET TEXT

Boolean Gadgets often have Intuitext strings rendered into them. Many file name requesters use Boolean Gadgets to display the choices of file names. Although the user cannot enter text into them, they can look much like a String Gadget. The point to remember here is that if you want to scroll a list of strings through a column of Boolean Gadgets, when the new string is written into the Gadget, the end of the old string may still be trailing off the end. One way to correct this is to write a string of blanks (length equal to the longest string) into the Gadget before you write the new string. It is easy to forget this, since doing the same type of scrolling with String Gadgets clears any extra characters from the end for you.

continued...

In the example program below, I write the word YES into the first Gadget. When you click on the Gadget, the work NO is written into the Gadget. If we did not write blanks to cover the 'S' in YES, when we wrote NO, we would have ended up with NOS. Admittedly, this is just one simple way to blank out the extra characters. The point is, remember they are there, and get rid of them somehow.

A bug fix in V1.2 causes GadgetText to be redrawn after an alternate image is rendered. Previously, if your Gadget had an alternate image as the highlight mode (GADGHIMAGE), when this new image was drawn, the GadgetText was not redrawn over it. The text was therefore lost. In V1.2 this has been fixed and the text is redrawn.

Verion 1.2 AmigaDos Enhancements

Masked Gadgets

Up until V1.2, a Gadget with the GadgetType of BOOLGADGET, should have had its SpecialInfo member point to NULL. This field was ignored for Boolean Gadgets. However, V1.2 has included the addition of masked Boolean Gadgets. Formerly, Boolean Gadgets could only be rectangles. Even if you specified an image to be rendered for the Gadget, the surrounding rectangle still defined the select box or area that caused Intuition to acknowledge that you clicked the Gadget. With the use of masked Gadgets, you can now specify a shape that redefines the select box of the Gadget.

This is a powerful enhancement that is only briefly touched upon in the 'Amiga Enhancer Software' manual.

The following steps are required to mask a Boolean Gadget.

1. Define the mask as a single bit-plane.

2. Define a BoolInfo structure, with its Mask member pointing to this data.

3. Set the SpecialInfo member of the Gadget to point to the BoolInfo structure.

4. Set the BOOLEXTEND flag for the Gadget. This tells Intuition that SpecialInfo points to a BoolInfo structure.

The BoolInfo structure is defined as follows:

```
struct BoolInfo
{   USHORT Flags;       /* defined below */
    UWORD  *Mask;
           /* bit mask for highlighting & selecting
            * mask must follow the same rules as an Image
            * plane.  It's width and height are determined
            * by the width and height of the Gadget's
            * select box. (i.e. Gadget.Width and .Height).
            */
    ULONG  Reserved;     /* set to 0     */
};
/* Set BoolInfo.Flags to this flag bit.
 * Future, additional bits might mean more stuff hanging
 * off of BoolInfo.Reserved. */
#define BOOLMASK 0x0001 /* extension is for masked Gadget */
```

Without a mask you can only specify a height and width for your Gadget. Therefore, a rectangle shape is the only option. If, however, we start with a rectangle, then mask out parts of it, we can produce other shapes.

In the example program below I define an image for the Gadget. This is a sort of oval in color 2, with a smaller oval inside of it in color 3. The mask bit-plane is a copy of the small oval. The effect is the following:

The oval-in-an-oval image is displayed as the Gadget. Clicking anywhere outside of the inner oval is not recognized by Intuition. If the user clicks in the inner oval, Intuition will acknowledge it and send you a GADGETUP/DOWN message if you have requested one. Also, only the inner oval will be highlighted if you have chosen the GADGHCOMP flag.

Although this is a very useful feature, beware of two problems. Neither the image nor the ghosting are rendered through the mask. Therefore, you cannot expect to be able to place two ovals right up against each other. You must remember that enough room must exist between them to accomodate the rectangles that contain them. Also, when ghosting is rendered, it is done over the whole rectangle, not just the Image/Mask. (This is a sort of pseudo-Fuzzy for those of you following along from last month).

The example at the end of the section on Intuition in the Enhancer manual gives a very simple implementation of masking. If you are going through the trouble of creating a Masked Gadget, you probably have an image of at least 2 bit-planes (4 colors). The example has you point the Mask member of BoolInfo to the array of data for the Image. This will make the Mask equal to whatever is in the first plane of your Image. This makes sense if your Image is only ONE bit-plane. Once you have multiple bit-planes in your Image, pointing Mask to the same data as your Image probably does not make sense. The bits set in the planes that make up your Image, represent colors. The Mask, however, is a single plane of bits that are either on or off. If you have a multi-colored Image, you probably want to create a separate array of data for the Mask that sets a bit wherever ANY plane of the Image has a bit set.

Of course, this is the obvious solution. In reality you are not obligated to make your Image and Mask correspond at all. The mask could be set to the outside of the oval, so the user would have to click in the space outside the oval to have Intuition acknowledge the click. In fact, the Masked Gadget is not required to even have an Image associated with it. You can define a mask on a plain-old-rectangular Gadget.

Cancel/Retry from the Keyboard

The Cancel and Retry Boolean Gadgets of system requesters can now be answered by left Amiga-V and left Amiga-B. This is a great help for the user who does not want to be tied to the mouse. This is made possible by the addition of a new Flag for Requesters. The NOISYREQ Flag means that normal input, such as keystrokes, will not be filtered as it usually is when the Requester is active. This

Flag is set on Requesters created by BuildSysRequest(). If you build a System Requester using BuildSysRequest(), you can listen for VANILLAKEY or RAWKEY, and associate keyboard input with the Boolean Gadgets of the Requester. If the user enters the appropriate keys, you can call FreeSysRequest() and remove the Requester without ever requiring mouse input from the user.

Now for some general Gadget Goodies...

Last time I mentioned that calling OnGadget() and OffGadget() can sometimes cause excessive flashing. This is because these routines call RefreshGadgets(). RefreshGadgets() refreshes all Gadgets in a list, allowing you to specify only the start of the list, not the end. This means that, assuming you do not do gymnastics to reorder your Gadgets each time you refresh, if you specify a Gadget near the start of your list, RefreshGadgets() re-renders most of the rest of your Gadgets even if they don't need it. Furthermore, if the Gadget specified is a Requester Gadget, ALL Gadgets in the Requester are redrawn. Even with the new less -flicker damage control of V1.2, this can mean flickering.

Version 1.2 offers RefreshGList() as an alternative. RefreshGList() lets you specify how many Gadgets to refresh (ie. the end of the list). This way you can refresh just the one Gadget that you modified. To go along with this idea AddGList() and RemoveGList() are also provided. These also let you specify the number of Gadgets to add/remove. Their parameters are shown below:

AddGList(*window_ptr, gadget_ptr, position, number_of_gadgets, requester_ptr*)

RemoveGList(*window_ptr, gadget_ptr, number_of_gadgets*)

RefreshGList(*gadget_ptr, window_ptr, requester_ptr, number_of_gadgets*)

Notice that AddGList() allows you to specify the requester to which the Gadget belongs. AddGadget() did not include this parameter. Therefore, when adding a Gadget to a requester, the new AddGList() should be used.

Listing One is a sample C program that opens a window with 5 Boolean Gadgets on it.

1. A Hit Gadget with GadgetText.
2. A Toggle Gadget with an alternate Image and GadgetText.
3. A Masked Toggle Gadget with optional ghosting. The ghosting is activated by a menu choice.

4-5. A Mutual Exclude pair. Each Gadget has an Image.

About the author

Harriet Maybeck Tolly owns a software company in Wilmington, Massachusetts called TollySoft. She, her husband Bob, and Max the dog, are currently specializing in Amiga software. She can be reached at:

PeopleLink: TollySoft

BIX: rtolly

Listing One

```
/*********************************************************
  Example "C" program showing use of Boolean Gadgets.

  This was compiled using Manx, Aztec "C", AmigaDos V1.2
  The code depends on V1.2 functions to operate correctly.
  It is intended to be run from CLI.

  Note: All Image data must reside in CHIP memory ( lower
  512K). If using Lattice, run ATOM on the compiled output
  before linking. If using Manx, use the +Cdb switch on
  the linker.

  Copyright (C) 1987 H. Maybeck Tolly, TollySoft

  This program is in the public domain and may be
  distributed free of charge.

*********************************************************/


#include <exec/types.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <functions.h>

struct IntuitionBase *IntuitionBase =0L;
struct GfxBase *GfxBase = 0L;
struct Window *ControlWindow = NULL;
struct IntuiMessage *MyIntuiMessage;


/*****************************************/
/* Border for toggle gadget (BoolGadget[1]) */
/*****************************************/

WORD BoolVectors[] = {0, 0,  51, 0,  51, 41,  0, 41,  0,0};
struct Border BoolBorder = {
   -1, -1,            /* initial offsets, gadget relative*/
   3, 2, JAM1,        /* pens (fore, back) and drawmode */
   5,                 /* number of vectors */
   (SHORT *)BoolVectors, /* pointer to the actual array */
                         /* of vectors */
   NULL                  /* no next border */
};

/* All image data must reside in CHIP memory. */

/*****************************************************/
/* Two Images for hit Gadget with altername images     */
/* (BoolGadget[2])                                     */
/*****************************************************/

/* Color 3 'X', 2 planes */

UWORD imageone[] = {
        0xFFF0, 0x0000, 0x0000, 0x0FFF,
        0x000F, 0xFF00,  0x00FF, 0xF000,
```

continued...

```
        0x0000, 0xFFF0,  0x0FFF, 0x0000,
        0x0000, 0x00FF,  0xFF00, 0x0000,
        0x0000, 0x000F,  0xF000, 0x0000,
        0x0000, 0x000F, 0xF000, 0x0000,
        0x0000, 0x00FF,  0xFF00, 0x0000,
        0x0000, 0xFFF0,  0x0FFF, 0x0000,
        0x000F, 0xFF00,  0x00FF, 0xF000,
        0xFFF0, 0x0000,  0x0000, 0x0FFF,

        0xFFF0, 0x0000, 0x0000, 0x0FFF,
        0x000F, 0xFF00,  0x00FF, 0xF000,
        0x0000, 0xFFF0,  0x0FFF, 0x0000,
        0x0000, 0x00FF,  0xFF00, 0x0000,
        0x0000, 0x000F,  0xF000, 0x0000,
        0x0000, 0x000F, 0xF000, 0x0000,
        0x0000, 0x00FF,  0xFF00, 0x0000,
        0x0000, 0xFFF0,  0x0FFF, 0x0000,
        0x000F, 0xFF00,  0x00FF, 0xF000,
        0xFFF0, 0x0000,  0x0000, 0x0FFF
        };

/* Color 2  '+', 2 planes */

UWORD imagetwo[] = {
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,
        0x0000, 0x0000, 0x0000, 0x0000,

        0x0000, 0x0FFF,  0xFFF0, 0x0000,
        0x0000, 0x0FFF,  0xFFF0, 0x0000,
        0x0000, 0x0FFF,  0xFFF0, 0x0000,
        0xFFFF, 0xFFFF,  0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
        0xFFFF, 0xFFFF,  0xFFFF, 0xFFFF,
        0x0000, 0x0FFF,  0xFFF0, 0x0000,
        0x0000, 0x0FFF,  0xFFF0, 0x0000,
        0x0000, 0x0FFF,  0xFFF0, 0x0000
        };

struct Image BoolImage = {
 0,0,64,10,2,
 imageone,
 0x03, 0x0, NULL
 };

struct Image BoolHImage = {
 0,0,64,10,2,
 imagetwo,
 0x03, 0x0, NULL
 };

/*****************************************/
/* Image for masked gadget (BoolGadget[3]) */
/*****************************************/

USHORT gadget_imageData[] =
{

0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0xFFFF,0xFFFF,0x0000,
0x0000,0xFFFF,0xFFFF,0x0000,
0x000F,0xFFFF,0xFFFF,0xF000,
0x000F,0xFFFF,0xFFFF,0xF000,

0x000F,0xFFFF,0xFFFF,0xF000,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x0FFF,0xFFFF,0xFFFF,0xFFF0,
0x0FFF,0xFFFF,0xFFFF,0xFFF0,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x00FF,0xFFFF,0xFFFF,0xFF00,
0x000F,0xFFFF,0xFFFF,0xF000,
0x000F,0xFFFF,0xFFFF,0xF000,
0x000F,0xFFFF,0xFFFF,0xF000,
0x0000,0xFFFF,0xFFFF,0x0000,
0x0000,0xFFFF,0xFFFF,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,

0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000

};

USHORT bool_mask[] =
{

0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x0FFF,0xFFF0,0x0000,
0x0000,0x00FF,0xFF00,0x0000,
```

```
0x0000,0x00FF,0xFF00,0x0000,
0x0000,0x000F,0xF000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000,
0x0000,0x0000,0x0000,0x0000
};

struct Image BoolMaskImage = {
    0,0,                /* LeftEdge, TopEdge */
    64,30,2,            /* Width, Height, Depth */
    &gadget_imageData[0],
    0x03,0x00,          /* PlanePick, PlaneOnOff */
    NULL,               /* Pointer to next Image */
};

struct BoolInfo BoolInfo = {
    BOOLMASK,          /* This is currently the only */
                       /* flag for boolean gadgets */
    &bool_mask[0],    /* Pointer to the mask */
    NULL
};

   struct IntuiText IntuiStr_blank =
        {2,3,JAM2,5,5,NULL,(UBYTE *)"    ",NULL};

   struct IntuiText IntuiStr_yes =
        {2,3,JAM2,5,5,NULL,(UBYTE *)"YES",NULL};

   struct IntuiText IntuiStr_no =
        {2,3,JAM2,5,5,NULL,(UBYTE *)"NO",NULL};

/*******************************************************/
/* Gadget structures for three example gadgets         */
/*******************************************************/

struct Gadget BoolGadget[5] = {
{

/* Hit gadget with Border */
   NULL,              /* pointer to Next Gadget */
   10,20,50,40,       /* (Left Top Width Height) Hit Box*/
   GADGHCOMP ,        /* Flags */
   GADGIMMEDIATE | RELVERIFY ,   /* Activation flags */
   BOOLGADGET,         /* Type */
   (APTR)&BoolBorder, /* pointer to Border Image */
   NULL,               /* no pointer to SelectRender */
   &IntuiStr_yes,      /* pointer to GadgetText */
   0,                  /* no MutualExclude */
   NULL,               /* pointer to SpecialInfo */
   0,                  /* no ID */
   NULL                /* no pointer to special data */
},

/* Alternate Image gadget */
   {&BoolGadget[0],110,20,64,10,GADGHIMAGE | GADGIMAGE ,
      GADGIMMEDIATE | RELVERIFY | TOGGLESELECT,BOOLGADGET,
      (APTR)&BoolImage,(APTR)&BoolHImage,&IntuiStr_yes,
      0,NULL,0,NULL},

/* This gadget has mask. It must have SpecialInfo point */
/* to a BoolInfo, and have the BOOLEXTEND flag set in   */
/* its activation flags.                                */

   {&BoolGadget[1],220,20,64,30,GADGHCOMP | GADGIMAGE  ,
     GADGIMMEDIATE | RELVERIFY | TOGGLESELECT | BOOLEXTEND,
     BOOLGADGET,(APTR)&BoolMaskImage,NULL,NULL,0,
     (APTR)&BoolInfo,0,NULL},

/* Mutual Exclude pair */
   {&BoolGadget[2],10,120,64,10,GADGHCOMP | GADGIMAGE |
        SELECTED,GADGIMMEDIATE,
BOOLGADGET,(APTR)&BoolImage,
        NULL,NULL,0,NULL,0,NULL},
   {&BoolGadget[3],110,120,64,10,GADGHCOMP | GADGIMAGE  ,
        GADGIMMEDIATE, BOOLGADGET ,(APTR)&BoolHImage,NULL,
        NULL,0,NULL,0,NULL}

};

struct NewWindow NewControlWindow = {
   20, 20,              /* start LeftEdge, TopEdge */
   320, 160,            /* start Width, Height */
   2, 3,                /* DetailPen, BlockPen */
                        /* IDCMP FLAGS */
   GADGETUP | GADGETDOWN | CLOSEWINDOW | MENUPICK,
                        /* Flags */
   WINDOWDRAG | WINDOWDEPTH | WINDOWCLOSE | ACTIVATE,
   &BoolGadget[4],      /* Pointer to FirstGadget */
   NULL,                /* no pointer to first CheckMark*/
   (UBYTE *)"Boolean Gadgets",/* Title */
   NULL,                /* no Pointer to Screen */
   NULL,                /* no Pointer to BitMap */
   20, 20,              /* Min size (no size allowed) */
   320, 160,            /* Max size (no size allowed) */
   WBENCHSCREEN         /* Type of screen */
};

/*******************************************************/
/* Menu declarations                                   */
/*******************************************************/

   struct IntuiText ghost_text = {
/* Frontpen, Backpen, Draw Mode   */
        2,1,JAM2,
/* Left and Top offsets, Font     */
        5,1,NULL,
/* Text to display,next IntuiText */
        (UBYTE *)"GHOST",NULL  } ;

   struct MenuItem ghost_item = {
/* Next item, left,top,width,height */
        NULL,0,5,140,10,
        HIGHCOMP | ITEMENABLED | ITEMTEXT | COMMSEQ,
/* MutualExclude, ItemFill (text)    */
        NULL, (APTR) &ghost_text,
/* SelectFill, Command, Subitem      */
        NULL, 'G', NULL } ;

   struct Menu gadget_menu = {
/* Next menu, left,top,width,height */
        NULL, 0, 0, 60, 10,
/* Flags, text                      */
        MENUENABLED, "Gadgets",
/* Pointer to first menu item       */
        &ghost_item } ;

/*******************************************************/
/* Main program                                        */
/*******************************************************/

main()
{
   struct MenuItem *ItemAddress();
   ULONG Signals, MIClass, MICode, itemnum;
   APTR MIAddress;
   LONG gad_pos, real_pos;

/* Open libraries */

   if (!(IntuitionBase = (struct IntuitionBase *)
    OpenLibrary("intuition.library",
                (LONG)LIBRARY_VERSION)))
   {
      printf("Can't open the intuition library\n");
```

continued...

```
      MyCleanup();
      exit(FALSE);
   }

   if (!(GfxBase = (struct GfxBase *)
    OpenLibrary("graphics.library",
                (LONG)LIBRARY_VERSION)))
   {
      printf("Can't open the graphics library\n");
      MyCleanup();
      exit(FALSE);
   }

/* Open window in which to display Boolean Gadgets. */

      if (!(ControlWindow =
         (struct Window *)OpenWindow(&NewControlWindow)))
      {
         printf("Couldn't open the control window.\n");
         MyCleanup();
         exit(FALSE);
      }

   SetMenuStrip(ControlWindow, &gadget_menu);

/* Loop forever until user clicks Close Gadget on window.*/

   for (;;) {  /* wait for a signal and process it */

      Signals = Wait(1L <<
        ControlWindow->UserPort->mp_SigBit);

         /* Process the Intuition message. */
         while (MyIntuiMessage=(struct IntuiMessage *)
                      GetMsg(ControlWindow->UserPort))
         {
         /* Get all the needed info and reply to message.
*/
         MIClass = MyIntuiMessage->Class;
         MICode = MyIntuiMessage->Code;
         MIAddress = MyIntuiMessage->IAddress;
         ReplyMsg(MyIntuiMessage);
         /* Determine what the message was. */

           switch (MIClass)
           {
            case MENUPICK:

            while (MICode != MENUNULL)
            {
              itemnum = ITEMNUM(MICode);
              switch(itemnum)
              {
/* User chose to ghost Masked Gadget. */
                  case 0:

/* Remove the Gadget from Intuition's control */
/* before we change any values.                    */
                  gad_pos = RemoveGList(ControlWindow,
                         &BoolGadget[2],1L);

/* Ghost/Unghost the Masked Gadget. */
                  BoolGadget[2].Flags ^= GADGDISABLED;

/* Return the Gadget to Intuition's control. */
                   real_pos = AddGList(ControlWindow,
                         &BoolGadget[2], gad_pos, 1L,
                         (LONG)NULL);

/* Refresh the display of the Gadget. */
                  RefreshGList(&BoolGadget[2],
                         ControlWindow, (LONG)NULL,1L);

                  break;

               } /* switch */

/* Get next Code if user extend-selected menu items. */
                  MICode = (ItemAddress(&gadget_menu,MICode))
                                 ->NextSelect;

               } /* while */
               break;

               case GADGETUP:

/* Check for GADGETUP on hit Gadget with IntuiText. */
               if (MIAddress == &BoolGadget[0])
               {

/* Write out a blank string to cover up old characters. */

/* Remove the Gadget from Intuition's control */
/* before we change any values.                    */
               gad_pos = RemoveGList(ControlWindow,
                  &BoolGadget[0],1L);

               BoolGadget[0].GadgetText = &IntuiStr_blank;

/* Return the Gadget to Intuition's control. */
               real_pos = AddGList(ControlWindow,
                  &BoolGadget[0],  gad_pos, 1L,
(LONG)NULL);

/* Refresh the display of the Gadget. */
               RefreshGList(&BoolGadget[0], ControlWindow,
                  (LONG)NULL,1L);

/* Now write out the new string. */

/* Remove the Gadget from Intuition's control */
/* before we change any values.                    */
               gad_pos = RemoveGList(ControlWindow,
                  &BoolGadget[0],1L);

               BoolGadget[0].GadgetText = &IntuiStr_no;

/* Return the Gadget to Intuition's control. */
               real_pos = AddGList(ControlWindow,
                  &BoolGadget[0],gad_pos, 1L, (LONG)NULL);

/* Refresh the display of the Gadget. */
               RefreshGList(&BoolGadget[0], ControlWindow,
                  (LONG)NULL,1L);
               }

               break;

               case GADGETDOWN:

/* Check for GADGETDOWN on Mutual Exclude Gadgets. */
               if (MIAddress == &BoolGadget[3] || MIAddress
                           == &BoolGadget[4])
               {
               gad_pos = RemoveGList(ControlWindow,
                  &BoolGadget[4], 2L);

               BoolGadget[3].Flags ^= SELECTED;
               BoolGadget[4].Flags ^= SELECTED;

               AddGList(ControlWindow, &BoolGadget[4],
                  gad_pos, 2L, NULL);
               RefreshGList(&BoolGadget[4], ControlWindow,
                  (LONG)NULL,2L);
               }

               break;

/* User clicked close Gadget. */
               case CLOSEWINDOW:
```

```
/* Reply to any outstanding messages. */
            while (MyIntuiMessage =(struct IntuiMessage *)
                 GetMsg(ControlWindow->UserPort))
                   ReplyMsg(MyIntuiMessage);
            MyCleanup();
            exit(TRUE);
            break;

          } /* switch */
        } /* while */
      } /* for */
   } /* main */

MyCleanup()
{
   if (ControlWindow) CloseWindow(ControlWindow);
   if (GfxBase) CloseLibrary(GfxBase);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
}
```

•AC•

Forth!

by Jon Bryan

...there are some fundamental differences between Multi-Forth and JForth.

In my last column, I mentioned that I had received a copy of JForth from Delta Research in Palo Alto. I am pleased to say that it has the makings of a good development system. I have already received an update, including the "real" manual which has been from preliminary version. Delta Research has been very helpful when I have called them with questions; from early indications, the support promises to be excellent.

While Delta Research has been preparing JForth, Creative Solutions has been busy too, working on improvements to Multi-Forth. I received a preliminary version of an on-line "help" disk from them. A Programmer's Toolbox is in the works, and the next revision (1.2) will probably be out by the time you read this article. Creative Solutions continues to improve an already robust product and provide outstanding support.

A CLASSIC COMPROMISE

Perhaps, it's East Coast versus West Coast (I'm being facetious). In any case, there are some fundamental differences between Multi-Forth and JForth. I would like to use this month's column to compare the two approaches, present a few benchmarks, and make some observations about the relative merits of each.

Foremost among the differences is the threading scheme each uses. Multi-Forth is a more traditional implementation of Forth. It is a "threaded interpretive" language, in the sense that it compiles data elements rather than machine instructions. In Multi-Forth's scheme, a word is compiled as a sixteen-bit "token" which represents an offset from a base register. On the other hand, JForth comes closer to the traditional definition of a "compiled" language because it lays down 68000 machine code. It's still "threaded," but by means of subroutine calls.

As a result, what you have is the classic trade-off between size and speed. Because each reference to a word only takes sixteen bits, Multi-Forth is quite small. I was able to squeeze the bouncing ball demo (presented a few issues back) down to about 34k. 25k might be attainable. JForth weighs in at about twice the size of Multi-Forth in its minimum configuration, and grows much faster, but runs certain kinds of tasks two to five times faster than Multi-Forth. Because so much time is absorbed by Amiga ROM, calls the difference in speed should be less in "real" applications. I would still expect JForth to be faster by a significant margin.

RELATIVE MERITS

I have always been impressed with the quality of Multi-Forth. It is the product of Creative Solutions' years of experience working with every conceivable 68000 system. The Amiga provided the testbed for a completely new kernal which was subsequently ported to the Atari and Macintosh. The product does show a few rough edges because of the brand-new implementation. Most of the bugs seem to have been worked out, though,and the next update will incorporate local multi-tasking. CSI has reduced the price from $180 to $90, making an already excellent value even better.

JForth is taking some getting used to. Its designers have taken an approach which differs considerably from Creative Solutions' approach. I've had to make a few mental adjustments in order to make the transition. Forth tends to amplify the differences as well. Everyone has his/her own concept of the right way to do things, and Forth allows each to be pursued with equal ease.

I have to say that I'm tremendously impressed by the amount of effort which has gone into JForth. Much of the kernal is included on disk in source form, allowing users to customize their own systems. Printing out all the source listings included on the two disks resulted in a two-inch-thick stack of paper! For $100, you get a lot for your money!

Both Multi-Forth and JForth are on the cutting edge of Forth technology. Multi-Forth uses what is perhaps the fastest, most compact form of token threading possible on the 68000 and local variables. The next revision should incorporate local multi-tasking. JForth uses subroutine threading for speed (with a heavy penalty in object code size), a different method of implementing local variables. JForth should also have local multi-tasking soon.

A COMPARISON OF FEATURES

JForth is loaded with nice touches. One particularly useful feature is a Motorola syntax disassembler invoked by the word DEF. As an example of its use, consider the word 2*, a fast way of multiplying a number by two. The response to "DEF 2*" is:

```
8E2  ADD.L  TOS,TOS     DE87     ( 6:  6:  0)
8E4  RTS                4E75     (16: 22:  3)
```

2* is 4 bytes long (1 cells), defined as 'both', indicating that 2* simply adds the top stack value (TOS) to itself. The first column is the relative address of the word, which may be different for you. The second column is the disassembly of

continued...

the instruction. The third column gives the hex values for the instruction and operand(s). The three values in parentheses indicate the number of CPU cycles used by the instruction, the running total of CPU cycles, and that total converted to microseconds. There is some guesswork involved, and occasionally, a question mark will be appended indicating that the disassembler couldn't give the exact number of cycles. This tip should be handy if you're a clock-watcher.

In the comment, a "cell" in JForth is four bytes. "Defined as 'both'" indicates that the word will be compiled in-line or called, depending on the value of a variable, MAX-INLINE. This option allows the user to have some control over the size of the object code. In fact, in a particularly time-critical application, MAX-INLINE could be set temporarily to a large value for a single definition and then reset back to the original value. The entire application doesn't have to suffer the size penalty if the extra time spent in branches and jumps can be tolerated. The word 2* can be used to define 4* like this:

```
: 4*   ( n --- 4n )   2* 2* both ;
```

Which when DEFed would appear as:

```
1AF38  ADD.L  TOS,TOS     DE87     ( 6:  6:  0)
1AF3A  ADD.L  TOS,TOS     DE87     ( 6: 12:  1)
1AF3C  RTS                4E75     (16: 22:  3)
```

4* is 6 bytes long (1 cells), defined as 'both'. I find the use of "both" curious. It seems that it should be used like "immediate" and follow the semicolon, but it apparently has something to do with a test which checks to see if the word can safely be used in-line. If the compiler decides that the word can't be used, it issues a warning and flags the word as "called."

Another feature of JForth is the mechanism used for calling libraries. Each library has a "formal definition" file which consists of entries for each function in the library. The entry for WritePixel in the graphics_lib.fd file is:

```
WritePixel(rastPort,x,y)(A1,D0/D1)
```

This entry indicates that the function expects a rastPort address and an x and y value and passes them in the A1, D0 and D1 registers. The syntax for calling WritePixel is:

```
call graphics_lib WritePixel
```

This syntax must be used within a definition. Of course, the appropriate values must be on the stack. "Call" would search the "graphics_lib.fd" file for the entry for WritePixel and then compile the instruction sequence to load to the function, the appropriate registers from the stack and jump to the function. JForth library calls always return the D0 register as well, so in the case of WritePixel, the call should properly be followed by "drop." I'll give a real code example shortly.

JForth, like Multi-Forth, includes words for defining the equivalent of 'C' data structures. Considering the Amiga's reliance on C-style data structures, they had very little choice. They also provide a program which converts C include files to JForth syntax, a utility which is sorely lacking in Multi-Forth. An example of a JForth structure definition would be:

```
:STRUCT Node
      APTR ln_Succ
      APTR ln_Pred
      BYTE ln_Type
      BYTE ln_Pri
      APTR ln_Name
  ;STRUCT
```

The fundamental difference between JForth structures and Multi-Forth structures is that members of JForth structures are typed, where those in Multi-Forth are not. Special words named ..@ and ..!, which automatically use the appropriate @ or ! word, are provided for accessing members of a structure. In JForth, LONG or APTR members will use @ and !, SHORT members W@ and W!, and BYTE members C@ and C!. An example of the syntax would be:

```
NT_MSGPORT myNode ..! ln_Type
```

I think this syntax is awkward and confusing. The fact that ..! has to look ahead in the input stream is contrary to standard practice. A better syntax might have had the member leave a flag and name the operators ?@ and ?! (don't you love these cryptic Forth names?). The concept certainly has merit, though.

Local variables are a recent development in Forth. My first exposure to these variables was in Amiga Multi-Forth. I think it's a great concept ! Delta Research must think so too, because they implemented them in JForth. The only problem is that they implemented them differently. In Multi-Forth they are specified using the syntax:

```
LOCALS| c b a |
```

Whereas in JForth the syntax is:

```
{ a b c --- }
```

The JForth syntax looks like a stack comment with braces substituted for parentheses. Multi-Forth locals are taken off the stack in the order in which they appear. JForth locals are called out in stack order, with the top stack item on the right. Multi-Forth allows any legal name to be used for a local, while JForth disallows a leading minus sign. BUT ... JForth gives you the source code for their implementation, so that this quirk can (and probably will) be fixed.

JForth locals include a few more bells and whistles as well. The notation:

```
{ a b | c --> c }
```

This notation allocates an uninitialized entry named "c" on the stack and causes the value of c to be returned. A few other operators are included. An example from the JForth manual reads:

```
: EXAMPLE   { a b | c --> c }
     a b + -> c  a c + -> c ;
```

The "->" is equivalent to "to" in Multi-Forth, and the syntax:

```
a b + -> c
```

The syntax stores the result of the addition of a and b into c.

Locals in both languages are self-fetching, which is not always the way you want things to be. The address of a local is accessed in Multi-Forth with the word "addr.of". JForth requires a different approach. In Multi-Forth, "addr.of" works immediately at the time a word is executed. The decision has to be made at compile time in JForth, using the directives "no@" and "yes@" to control how the local is handled. Because I was expecting behavior similar to Multi-Forth, this one caught me the first time. Again, here is an example from the JForth manual:

```
: EXAMPLE   { a b c --- sum+1 }   \ "sum+1" is a comment
     a b + c + -> c
     no@  ( self-fetching off )  1 c +!
     yes@  c ;
```

The example which will be given shortly in a circle algorithm should help to make this more clear.

The last difference between JForth and Multi-Forth local variables is that JForth locals are left on the data stack, while Multi-Forth moves them to the return stack (the approach used by another company is to move them to a third stack!). Each approach has its own quirks. In Multi-Forth, you must be careful with words like >R and R> when you're using locals. In JForth, you must watch what you're doing with the data stack. Both problems have bitten me. A completely separate stack has the fewest side-effects, but it is also the slowest.

FIERO OR TRANS-AM?

The biggest problem I see with JForth is the size. It starts off big and gets much bigger in a hurry. The system comes up without any graphics support, and adding the support absorbs over 24k. In contrast, Multi-Forth comes up with mostly equivalent graphics support and is half the size of JForth. By the time the demos are compiled on a 512k machine, there is only about 50k of memory left. There were times when I couldn't run Ed without shrinking some windows. If you're going to use JForth as a development system, I would recommend adding at least another 512k of memory. A full megabyte would be better, and you should pray that Delta Research gets their target compiler done soon.

If the target compiler works as promised by Delta Research, then a turnkey of a simple program like "Hello World" should compile to less than 1k. Without that utility, I'm not sure whether you can develop viable applications for 512k machines. Multi-Forth doesn't have a target compiler either, but they have less need for one, since they have a very compact implementation to begin with.

It's debatable how much difference between the two you will see in terms of speed on equivalent applications. JForth is faster, but it achieves that speed by making EVERYTHING fast and big. In Multi-Forth, you would typically optimize only the parts that needed the extra speed. In most applications, that probably wouldn't even be necessary. As both a simple benchmark and an example of JForth code, I have included a version of Bresenham's circle-drawing algorithm. This example illustrates some of the differences between the two implementations. Of particular interest is JForth's handling of local variables and the mechanism for calling library routines. In timing comparisons, JForth is approximately 35% faster than Multi-Forth on this particular algorithm. Each algorithm is presented in the form necessary to run from the default configuration of each language. To try them out bring up the language, compile the code, and then execute "samplewindow" to open a window to draw into. When you finish, "closecurrentwindow" will clean things up. Enjoy.

SOURCE CODE

First, the algorithm in JForth:

```
\ Bresenham/Michener circle algorithm.
\ Written in JForth by Delta Research
\ From "Fundamentals of Interactive Computer Graphics"
\ by Foley and Van Dam.
\ Jon R. Bryan 3/22/87

INCLUDE? GR.INIT JU:AMIGA_GRAPH
GR.INIT

INCLUDE? TASK-LOCALS JU:LOCALS


: 4*   ( n -- n*4 )   2* 2* both ;


newwindow my-window


: samplewindow   ( -- )
   my-window newwindow.setup
   my-window gr.openwindow
   gr.set.curwindow ;


: closecurrentwindow   ( -- )   gr.closecurw  gr.term ;


variable xoffset
variable yoffset


: >xyoffset   ( x\y -- xrelative\yrelative )
   swap xoffset @ +  swap yoffset @ + ;


: dot   ( x\y -- plots a pixel )
   >xyoffset  gr-currport @ rot rot
   call graphics_lib WritePixel drop ;


: circle_points   ( x\y -- )
   over negate over negate
   { x y x- y-  --- }   \ remember, no preceding -
   x y  dot  y  x dot  x- y  dot  y  x- dot
   x y- dot  y- x dot  x- y- dot  y- x- dot ;
decimal
```

continued...

```
: mich_circle   ( xcenter\ycenter\radius -- )
   3 over 2* -  0
   { xcenter ycenter y d x --- }
   \ xcenter and ycenter must be in the local list
   xcenter xoffset !  ycenter yoffset !
   BEGIN   x y <
   WHILE   x y circle_points
     d 0<
     IF  x 4*  6 + no@ d +!
     ELSE  yes@
       x y - 4*  10 + no@ d +!
       -1 y +!
     THEN
     1 x +!
   REPEAT  yes@
   x y =
   IF  x y circle_points  THEN ;
```

The equivalent in Multi-Forth

```
\ Bresenham/Michener circle algorithm
\ written in Multi-Forth by Creative Solutions
\ Jon R. Bryan 3-22-87


variable xoffset
variable yoffset


: >xyoffset   ( x\y -- xrelative\yrelative )
   swap xoffset @ +  swap yoffset @ + ;


: dot   ( x\y -- )
   >xyoffset Rport !a1 !d1 !d0 graphics 54 ;


: circle_points   ( x\y -- )   \ 8-way symmetry
   over negate over negate
   locals|  -y  -x  y  x  |
   x  y dot   y x dot  -x  y dot   y -x dot
   x -y dot  -y x dot  -x -y dot  -y -x dot ;


: mich_circle   ( xcenter\ycenter\radius -- )
   3 over 2* -  0
   locals|  x  d  y  |
   \ You can still access the data stack here
   xoffset !  yoffset !
   BEGIN   x y <
   WHILE   x y circle_points
     d 0<
     IF  x 4* 6+ addr.of d +!
     ELSE
       x y - 4* 10+ addr.of d +!
       -1 addr.of y +!
     THEN
     1 addr.of x +!
   REPEAT
   x y =
   IF  x y circle_points  THEN ;
```

•AC•

Programming in 68000 Assembly Language

by Chris Martin

Last month, we ended with a discussion of registers and addresses. The 68000 microprocessor contains on-chip memory locations called registers. There are two types of these 32-bit memory locations: data registers and address registers. The data registers are called D0, D1, D2, ... D7 and they can be used to store numbers up to 32-bits long (in decimal, up to 4,294,967,295). Address registers, although 32-bits long, can hold only numbers up to 24-bits in length (the number 16,777,216). These registers are called A0, A1, A2, ... A6 and are used primarily to store addresses in the computer's memory, much like street addresses in a telephone book.

Aside from data and address registers, the 68000 has some other registers.

Program Counter

Assembly programs are stored sequentially in the computer's memory. The PC (program counter) register is a 32-bit register; the first 24 bits contain the address of the program instruction that is currently being executed. The PC is automatically updated when the 68000 reaches the next instruction in the program or when it is told to branch to another part of the program, in a different section of the computer's memory.

Status Register

The SR (status register) is a 16-bit register which holds a number of bit-sized "flags" which indicate the current status of the 68000. These flags tell whether a computed value is positive or negative, or whether a compared value is greater than, equal to, or less than another value. We'll talk more about this useful register next month.

Stack Pointer

The SP (stack pointer) is address register A7, not mentioned above. A special area in memory, called the stack, is reserved for storing temporary data and variables. The SP points to the top of the stack - the point at which fresh data may be stored or old data removed. The stack can be very useful for storing numbers temporarily.

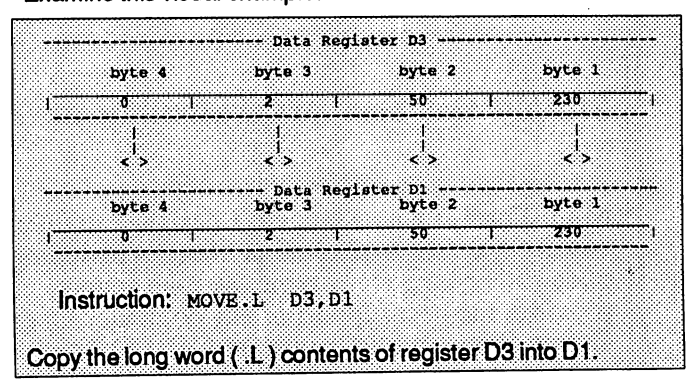
Addressing Modes

When writing a program in assembly language, there are several ways to access and move memory. These are called addressing modes. There are two main categories of addressing modes: memory addressing, in which memory is addressed; and register addressing, in which registers are addressed. During our discussion on addressing modes, we will use actual assembly instructions: MOVE, which is used to copy data from one place to another and ADD and SUB, which add and subtract.

Register Direct Addressing

Register Direct addressing is used in operations involving registers (address or data). For example, the contents of two registers may be added or subtracted, or the contents of one register may be transferred to another. When moving, adding or subtracting, the programmer may specify (when using certain instructions) the size of data to be used: a byte, word, or long word (8, 16, and 32 bits respectively).

Examine this visual example:



Notice that the contents of register D3 are copied to register D1. At the end of the MOVE command, we placed a .L suffix. This suffix indicated that the entire long word was to be copied. Likewise, a .B suffix indicates a byte and a .W indicates a word.

When dealing only with Register Direct Addressing, the destination register (where the data will be copied, added, or subtracted into) also must be taken into account. When the destination register is an address register (A0 to A6), you must add an "A" to the end of the instruction. For example, instead of the MOVE.L D3,D1 instruction, you must use the following instruction: `MOVEA.L D3,A4`

continued...

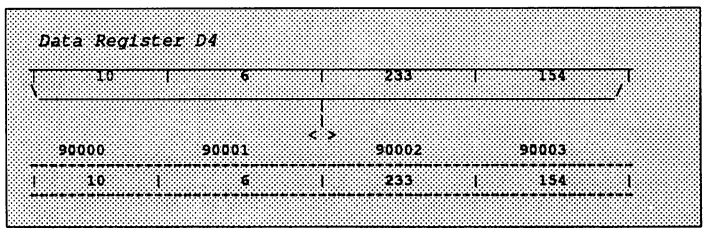
This instruction moves a long word from data register D3 to address register A4. Note that only 24 bits of the 32 bit word will be copied to the address.

Absolute Addressing

You can use the Absolute Addressing mode to copy a register's data into a memory location. For example, if we want to put the entire contents of register D4 into memory starting at location 90000. In this case, we use the instruction:

```
MOVE.L  D4,90000
```

Visually, this is what happens:



In this mode, as in every mode, we can specify the size of the data to be moved, added, etc by using the suffixes .B, .W, or .L . Other examples of this mode are:

```
MOVE.B   D5,10345
```

Copies the lowest byte of D5 into 10345;

```
MOVE.W   A3,2003
```

Copies the lowest word of A3 into 2003 and 2004 (remember, a word takes up 2 bytes).

Immediate Addressing

Take the following BASIC statement:

```
LET A = 10
```

This statement is an example of taking an Immediate value (10) and placing it into A. Here is an equivalent in assembly, where we will copy a value of 10 into register D6:

```
MOVE.B   #10,D6
```

Because the data registers (such as D6) are 4 bytes long, the 10 will be copied into the lowest byte; the remaining 3 bytes will be unaffected.

Another form of Immediate Addressing is called Quick Addressing. As the name implies, Quick Addressing allows the instructions to be performed faster than usual. On programs that require intensive calculation, speed is important to the user. In such programs, use of the quick addressing mode can have an appreciable effect on the overall running time.

Many instructions in 68000 assembly language have a "Q" added to the end of them. This "Q" indicates Quick Addressing. In the MOVEQ instruction, an 8-bit number can be quickly copied into a 32-bit data register. For example:

```
MOVEQ    #80,D2
```

This example copies 80 into D2 using quick addressing.

Address Register Indirect Addressing

This useful mode allows the programmer to access data in memory locations pointed to by address registers. Suppose that address register A5 contains 90000. Also suppose we want to copy the data stored in address 90000 into data register D3. Thus, we would use the statement:

```
MOVE.L   (A5),D3
```

This statement would indirectly copy the contents of 90000 (where A5 points to) into data register D3. Likewise, we could copy the contents of a data register into the memory location pointed to by an address register. For example:

```
MOVE.L   D3,(A5)
```

This statement copies the contents of D3 into the address in A5.

If we wanted to copy data from memory (pointed to by an address) to a data register, as in the instruction "MOVE.L (A5),D3", and at the same time, increment or decrement the address in A5, then we would use postincrement or predecrement. Here are some examples:

```
MOVE.L    (A5)+,D3
```

Copies the value in location pointed-to in A5, then increments A5 by 4 (a .L long word takes 4 bytes). If we used a .B, A5 would only be incremented by 1; if we used a .W, A5 would be incremented by 2.

```
MOVE.L    -(A5),D3
```

First the address register is decremented by the size (1, 2, or 4) and then the data is accessed. Hence, PREdecrement.

Address Register Indirect with Displacement

This mode is very versatile. You may access data from locations that are relative to a certain address. You determine the offset from a set location in the instruction. For example:

```
MOVE.B    3(A3),D2
```

This statement copies the data in the address, pointed to by A3 + 3, into the lowest byte of data register D2. In this mode, the range of the displacement is -32 K to +32 K.

Address Register Indirect with Index and Displacement

This mode is similar to the last, but it allows you to specify another offset with data or an address register. For example:

```
MOVE.B    24(A3,D4),D2
```

This statement copies the contents of the memory address, pointed to by A3 + D4 + 24, into the lowest byte of data register D2. We could modify the statement to read:

```
MOVE.B    24(A3,D4.L),D2
```

This modification changes the address pointed to into A3 + (the long word value of) D4 + 24. In any case, the displacement value (24 in the above cases) can range from -127 to +127. The index (A3 + D4 above) can range from -32 K to +32 K.

Well, that's all for this month. Next month, we will discuss the condition flags and hopefully get into some PROGRAMMING! In the mean time, please buy a good assembler for the Amiga (preferably the Commodore-Amiga Macro Assembler). Until next month....

•AC•

The AMICUS Network

by John Foust

There is a sense of nervous anticipation in the air these days among Amiga owners. The Amiga 500 and 2000 have made a splash in the formerly calm waters of Amiga Creek, and with the upcoming influx of Amiga owners, it may very well turn into Amiga Lake.

On my wall, I have a cartoon drawn by Jim Miller, a well-known computer music software programmer. Miller gave out these cartoons at last summer's National Association of Music Marketers Show in Chicago. He writes for the PC market and ports to no other computers. Whenever someone asked "Why haven't you ported to the ____ computer yet?," he would give them the cartoon. The cartoon looks like a map of a continent's shoreline, with a large river flowing from left to right, dumping into an ocean.

At left, a bump in the river is labeled "Lake Apple." It drains into the "IBM River," which is fed by both the "Atari Stream" and "Amiga Creek." The ocean at right takes up half the cartoon; it is labeled "IBM Ocean." The cartoon explains why Miller writes for the PC market. Perhaps Commodore should adopt this cartoon as the corporate reasoning behind the introduction of the Amiga 2000.

Commodore will be present at the summer Consumer Electronics Show in Chicago starting May 30, showing off the Amiga 500 and 2000 to hordes of electronics dealers. After this appearance, there will be a long drought of Amiga shows until the fall of 1987. I might very well be forced into writing about user groups and public domain software again, instead of Amiga shows.

AMICUS 17

AMICUS disk 17 is now ready. This is a telecommunications disk which contains six terminal programs: Comm version V1.33, ATerm V7.2, VT-100 V2.6, Amiga Kermit V4D(060), VTek V2.3.1 and AmigaHost 0.9. Comm is a robust terminal program with Xmodem transfers, as well as the WXmodem file transfer protocol implemented on People Link. This protocol can save quite a bit of time on file transfers. ATerm includes the Super Kermit transfer protocol for the Source network. VT-100 is Dave Wecker's well- known VT-100 emulator. It has Xmodem and Kermit protocols, as well as strong scripting abilities. Amiga Kermit is a port of the Unix C-Kermit, except for the DIAL and SCRIPT commands. VTek is a Tektronix graphics terminal emulator based on the VT-100 program V2.3. AmigaHost is designed for use on Compuserve. It includes RLE graphics abilities and the CIS-B file transfer protocol.

VTek also contains the latest version of the 'arc' file compression and library program. The full documentation for 'arc' is included, along with a tutorial on the basics of un-'arc'ing files.

The 'arcre' program is useful for making 'arc' files. This process is described later in this column with the files on Fish disk 53. The C source is included there also. 'FixHunk' is a program for people with expansion memory. It is described later with Fish disk 36 . 'FixObj' is described below on Fish disk 38.

'Txt' is a very handy tool for manipulating text files. Very often, if you visit bulletin boards that specialize in other computers, you encounter text files that will not load into 'ed'. They have strange line endings, or they appear to be double-spaced, or they contain unprintable binary characters, or paragraphs appear as one long sentence. The 'txt' program will filter these files into something readable on the Amiga. The C source is included, and has conditional compilation code for the MS-DOS world. It works like a charm on those machines, too.

Although unrelated to telecommunications, this disk also includes an executable version of 'addmem' for those people who built the memory expansion listed in *Amazing Computing*, Volume 2.3.

Fish disks 36 to 53

The Fred Fish collection of public domain disks continues to grow. It has been a long time since I have described the contents of the newest disks. In fact, my last description covered up to disk 35,and the collection is now up to disk 58. In order to cover all the disks from 36 to 53, I will skip some programs, and give only the highlights of each disk. A complete catalog of available disks is on page 96. I will cover new disks 54 to 58 next month.

Fred Fish 36

'Acp' is a copy program, much like the Unix 'cp' copy program. Why not use the AmigaDOS 'copy' command? There are several reasons. First, some people have a primal urge to make their home computer act just like the computer they used in college, or use everyday at work. Thus, this program uses Unix and MS-DOS style wildcards for naming files, such as '*.*'. Second, the AmigaDOS 'copy' program is not very fast, and a simple technique can be used to increase its speed.

continued...

'Echo' is a tiny program that prints a line of text to the screen. This program would normally be invoked from a script file, such as the 'startup-sequence' file. It has options for modifying the text with color and italics. This program was written by Larry Phillips, a co-SYSOP on Compuserve.

'FixHunk' is an absolute necessity for Amiga owners with expansion memory. Amiga program files are composed of 'hunks' of program data and 68000 code. All graphic image data must be placed in CHIP memory, while program code can reside in FAST memory. Each hunk within the program file has a tag which tells which type of memory this particular hunk prefers. Some programmers are not careful about setting this tag, which results in garbled screen displays for gadgets and program images. Fixhunk modifies the tags on hunks, and moves all data to CHIP memory. This process usually fixes programs that do not work in expansion memory. Fixhunk was written by Dan 'DJ' James, a co-SYSOP on People Link.

'KickBench' is a program to create a combined Kickstart and Workbench disk. This program was developed to allow an unattended reboot of an Amiga after a power failure, an important consideration for bulletin board systems. Canadian wizard Alonzo Gariepy disassembled the boot code on the Kickstart and Workbench disks and wrote a series of patches. The patches are designed for use with the 'disked' program from the developer disks, so you might have to borrow 'disked' from a developer friend.

Fred Fish 37 is a Little Smalltalk system. C source is included, along with many example Smalltalk programs.

Fred Fish 38

'FixObj' is useful for those people perplexed by 'file is not an object module' error messages in the CLI. There is a chance that this program file was transmitted via modem and the Xmodem protocol, which can introduce garbage characters at the ends of files. Fixobj removes these garbage characters.

'Window' is an example from Commodore, showing how to attatch a CLI window into a custom screen.

Fred Fish 39

'ansiecho' is another 'echo' program. This disk also includes assembly language source for it, along with other CLI commands such as 'cls', 'touch' and 'ask'.

This disk also has an example device driver. It functions as a simple RAM disk for demonstration purposes.

This disk has the first set of ray tracing images by Dave Wecker. Ray tracing is a technique for producing very life-like computer graphics. Many computer generated images look unrealistic because they are evenly illuminated, or contain no shadows, as if there was no light source in the picture. Ray traced pictures have a light source, and the objects in the picture often reflect or transmit light as well. Ray-traced pictures often contain glass or mirrored balls, because these objects are easy to model mathematically and because of the extremely life-like effect. Ray tracing involves following the path of the light rays that would reach the viewer's eye from the scene modelled in the computer's memory.

Instead of these images, you might want to get the pictures on Fish disk 44. On that disk, the pictures have been converted to IFF format and can be viewed by any IFF viewer program. On disk 39, the images are in a non-IFF format which requires a uniquely slow display program.

Fred Fish 40

This disk contains shareware and freeware programs. This means they are not in the public domain, and chances are, source code is not provided. Most of these programs are obsolete now, as the authors have since replaced them with newer versions. Such is the disadvantage of public domain and publicly distributed software; you rarely know if you are getting a debugged and most recent version.

Fred Fish 41

'AmigaVenture' is a full-fledged adventure writing system in AmigaBasic, written by Mitsuharu Hadeishi. With this system, you can create your own text-based adventures.

The 'GetFile' directory holds the source, object and executable of Charlie Heath's file requester, popularized in his TxED editor. This is an excellent example of an fast and easy-to-use file requester, as opposed to inflexible file requesters that enforce a ten- to twenty second wait before you can even tell the system to select a different disk. It has been over a year since Heath's requester appeared. Very few Amiga developers have incorporated it, and no one has improved upon it. It has been long enough. Amiga developers, please use it! All he wants is his name in your manual! (Just a little joke, Heath.)

'SetFont' allows you to change the font used in the CLI window. By entering 'setfont emerald 20', you can work in the CLI with the Emerald font.

Fred Fish 42 holds an Amiga version of MicroGNUEmacs, yet another Emacs clone for microcomputers. This one is intended to be completely compatible with the GNU Emacs. 'GNU' stands for "GNU's Not Unix." This version reveals an effort to produce a public domain version of the Unix operating system, as well as the programs traditionally found on Unix systems.

Fred Fish 43

'Copper' is a copper list disassembler by Scott Everndon. (The copper is part of a graphics chip in the Amiga used to control the display.) C source is included.

'PopColours' is a program to change the colors of any screen on the Amiga, using sliders in a movable window. It was written by two developers at Canada's *Transactor* magazine, which accounts for the spelling of the program's name.

Fred Fish 44

This disk contains some well-drawn icons, new material about the IFF standard from Commodore, the IFF versions of the Wecker ray tracing pictures on disk 39, and a program to view IFF ILBM files.

Fred Fish 45 contains the game of Clue in AmigaBasic, a version of 'make', and a few IFF pictures. 'Update' is a program to update a disk of programs. Files on the older disk will be replaced by similarly named, but newer files on the other disk. 'Wherels' searches a disk for a file of a given name.

Fred Fish 46

'Egad' is the long-awaited gadget editor from the Programmer's Network. This programmer tool edits menus and gadgets and generates C source code to recreate a particular screen.

'Jive' and 'ValSpeak' are filters used to transform texts. 'Jive' transforms text into street English, while 'ValSpeak' translates documents into Valley Girl English.

Fred Fish 47 contains the 3-D robot arm simulation and the Juggler demo, along with version 2.4 of the Wecker VT-100 terminal program, with Xmodem and Kermit file transfer protocols.

Fish 48

'MemWatch' is a program that watches for changes in low memory. Badly behaving programs often inadvertently change values stored in this area of memory, leading to crashes. This program restores the changed bytes and puts up a requester warning of the damage.

'Profiler' is an execution profiler for Manx C version 3.30e. It can identify which regions of a program are most often executed. Source is included.

Fish 49

'MultiDef' is a programmer tool used to scan a set of object modules and library files for multiply-defined symbols. 'MyUpdate' is a program to strip comments from C 'include' files. 'QMouse' checks whether or not the left mouse button is pressed. In a script file, it can return a failure code that can change the course of execution in the script file. For example, it could decide to copy programs to the RAM: disk, if the mouse button was pressed during the startup-sequence.

Fish 50

This disk contains more shareware programs. 'Asm' is a shareware macro assembler, compatible with the assembler described in the AmigaDOS manuals. This is not the most recent version, so you might want to check your user group's library for the newest. An older version is present on Fish 46.

This disk contains the games Missile and 3-D Breakout. Fish 50 also contains the SiliCon: CLI shell, an older version of the Perfect Sound sampled sound editor, and the source code to a Unix System V compatible 'arc' program.

Fish 51

'Bison' is the GNU project's replacement for Unix 'yacc', a programmer tool called a parser generator.

This disk has an update of Unix 'compress' from Fish disk 6. Fish 51 also includes 'DifSsed', portable versions of 'sq' and 'usq' file compression programs, and Unix-like 'diff' and 'sed', for finding differences between files, and creating new files from that list of differences.

continued...

Fish 52

This disk contains a replacement for the AmigaDOS 'assign' command (written in C), 'Tek4010', a Tektronix graphics terminal emulator, and 'VDraw', a shareware drawing program.

Fish 53

This disk contains some animations and a public domain player program for Aegis Animator.

'ARCre' is a program to rename files before using the 'arc' program. Unfortunately, the 'arc' file compression program maintains the MS DOS standard of only thirteen-character file names, while many Amiga files have longer names than that. 'ARCre' renames all the files in a directory and then creates a script file to rename the files back to their original names. When the user unpacks the 'arc' file, he enters 'execute execute.me' at the CLI prompt, and all the files are renamed.

There is also a public domain C compiler for the 68000 on this disk, but it is not completely ported to the Amiga yet. It produces assembly language source code files for output, which means you still need to assemble the output of the compiler. It does not create executable files. With the advent of public domain assemblers and linkers, it may be only months before a public domain Amiga programming environment is created.

'VC' is a simple, VisiCalc-like spreadsheet program. This program is an update of the version on Fish disk 36, but it still has a few bugs (like not being able to print the spreadsheet). Source code is included.

More Amiga shows

AmigaWorld magazine will sponsor an Amiga show, yet another show on the West coast, called Amiga Expo. It will be at Brooks Hall in San Francisco on September 11 through 13. Commodore is expected to be on hand. Hopefully, they will bring a booth and some Amiga machines, and maybe even some demonstrations with flashing lights and music. (Ah, my cynicism is showing...) This show is being coordinated by the same company that produces the Mac Expo shows. For more information, contact Chris Rassias at World Expo Headquarters at (617) 329-8334.

As noted in a previous column, the name of the October 10 Amiga show in New York has been changed from Amiga Expo to AmiExpo. For more information, call (800) 32-AMIGA; in New York state, call (212) 867-4663. The winter show has been moved to Los Angeles at the Airport Hilton on January 22 to 24, 1988. The summer 1988 show will be in Chicago at the Hyatt Regency July 22 to 24.

Apology

In *Amazing Computing* V2.3 which unveiled the Amiga 500 and 2000, I gloomily discussed the prospects of forming a national Amiga user group. I did not write to discredit the people working towards such an effort, I only wrote to explain why I did not want the AMICUS name on any other group. I did not intend to imply that the people working on a national Amiga group are going to run off to Jamaica with thousands of dollars.

People Link conferences

The Sunday night conferences on the People Link have become extremely popular. Attendance figures have jumped above those of conferences on other networks.

The Sunday night conference starts out with general rumor-swapping and news exchanging. Later, the group splits into developer and user group conferences. On Monday nights, the MIDI music group meets. The unofficial hostess of the music group is Peggy Herrington, whose name can be found above many Amiga articles in other magazines. At last word, a beginner's conference was in the works for a different night during the week. This conference is targeted at people who want to learn more about their Amigas.

Let me try to describe an unsupervised online conference. Imagine being at a party , in the dark,with eighty others . Every few seconds, someone in the room utters a sentence and everyone can hear it perfectly. Some people have

formed groups. One group might be talking about Amiga hardware, another might be telling jokes about a television show; yet it is all happening at the same time.

In the conference, whenever anyone types a line of text, the message is transmitted to every other participant in the conference. One person might ask a question and another might answer it a few lines later. The intervening messages might be talking about something else entirely.

Conferences are sometimes hard to follow. To the uninitiated, they are bewildering! Puns and jokes slip in. Talk is littered with telecom idiom - abbreviations such as "brb" for "Be right back," and ":-)" to indicate humor (that is a smiley face, on its side) and "imho" for "In my humble opinion." Occasional pearls of wisdom surface, too. After a heated discussion about the new Macintosh and IBM machines and the benefits of multitasking, Mike Scalora remarked "Yeah, it's a shame that less than 200,000 people know anything about computers." Scalora was talking about Amiga owners, of course.

Conferences can also be supervised, making them much easier to follow. This is often the case when a special guest is present. A moderator regulates who will ask questions and the conference guest answers them in turn.

RJ Mical discovered online conferencing late one Sunday night on People Link. Telecom denizens are often surprised to learn that they know more about telecommunications than a hot-shot programmer. Mical quickly adjusted and dove into the conversation headfirst.

Other recent conference guests include Dan Schein of Commodore West Chester's telecommunications department and Bob Page and Rich Miner of Zoxso, the people who once promised ZLI, and are now promising an Amiga expansion peripheral for image processing. Another visitor is Steve Grant from Access Associates, makers of the the Alegra memory expansion.

Grant will lend his Amiga hardware expertise to future issues of *Amazing Computing*, if all goes well. His first article will discuss Amiga expansion issues. Miner and Page had an article on Amiga Live! in the first issue of *Amazing Computing*. They hope to write again in the future. Mical hopes to write for us also, after he finishes his long-awaited game for Electronic Arts.

My favorite guests at the conferences are all of you, the readers of *Amazing Computing*. Many of you have visited the *Amazing Computing* section on People Link and left a messages just to say hello or thanks. Your comments about the magazine are taken to heart.

•AC•

The AMICUS & Fred Fish Public Domain Software Library

This software is collected from user groups and electronic bulletin boards around the nation. Each Amicus disk is nearly full, and is fully accessible from the Workbench. If source code is provided for any program, then the executable version is also present. This means that you don't need the C compiler to run these programs. An exception is granted for those programs only of use to people who own a C compiler.

The Fred Fish disk are collected by Mr. Fred Fish, a good and active friend of the Amiga.

Note: Each description line below may include something like 'S-O-E-D', which stands for 'source, object file, executable and documentation'. Any combination of these letters indicates what forms of the program are present. Basic programs are presented entirely in source code format.

AMICUS Disk 1

ABasic programs: Graphics

3DSolids	3d solids modeling program w/sample data files
Blocks	draws blocks
Cubes	draws cubes
Durer	draws pictures in the style of Durer
FScape	draws fractal landscapes
Hidden	3D drawing program, w/ hidden line removal
JPad	simple paint program
Optical	draw several optical illusions
PaintBox	simple paint program
Shuttle	draws the Shuttle in 3d wireframe
SpaceArt	graphics demo
Speaker	speech utility
Sphere	draws spheres
Spiral	draws color spirals
ThreeDee	3d function plots
Topography	artificial topography
Wheels	draws circle graphics
Xenos	draws fractal planet landscapes

ABasic programs: Tools

AddressBook	simple database program for addresses
CardFile	simple card file database program
Demo	multiwindow demo
KeyCodes	shows keycodes for a key you press
Menu	run many ABasic programs from a menu
MoreColors	way to get more colors on the screen at once, using aliasing
shapes	simple color shape designer SpeakIt speech and narrator demo

ABasic programs: Games

BrickOut	classic computer brick wall game
Othello	also known as 'go'
Saucer	simple shoot-em-up game
Spelling	simple talking spelling game
ToyBox	selectable graphics demo

ABasic programs: Sounds

Entertainer	plays that tune
HAL9000	pretends it's a real computer
Police	simple police siren sound
SugarPlum	plays "The Dance of the Sugarplum Fairies"

C programs:

ATerm	simple terminal program, S-E
cc	aid to compiling with Lattice C
decvnt	opposite of CONVERT for cross developers
Dotty	source code to the 'dotty window demo
echox	unix-style filename expansion, partial S, O-D
fasterfp	explains use of fast-floating point math
FixDate	fixes future dates on all files on a disk, S-E
freedraw	simple Workbench drawing program,S-E
GfxMem	graphic memory usage indicator, S-E
Grep	searches for a given string in a file, with documentation
ham	shows off the hold-and-modify method of color generation
IBM2Amiga	fast parallel cable transfers between an IBM and an Amiga
Mandel	Mandelbrot set program, S-E
moire	patterned graphic demo, S-E
objfix	makes Lattice C object file symbols visible to Wack, S-E
quick	quick sort strings routine
raw	example sample window I/O
setlace	turns on interlace mode, S-E
sparks	qix-type graphic demo, S-E

Other executable programs:

SpeechToy	speech demonstration
WhichFont	displays all available fonts

Texts:

68020	describes 68020 speedup board from CSA
Aliases	explains uses of the ASSIGN command
Bugs	known bug list in Lattice C 3.02
CLICard	reference card for AmigaDOS CLI
CLICommands	guide to using the CLI
Commands	shorter guide to AmigaDOS CLI commands
EdCommands	guide to the ED editor
Filenames	AmigaDOS filename wildcard conventions
HalfBright	explains rare graphics chips that can do more colors
ModemPins	description of the serial port pinout
RAMdisks	tips on setting up your RAM: disk
ROMWack	tips on using ROMWack
Sounds	explanation of the Instrument demo sound file format
Speed	refutation of the Amiga's CPU and custom chip speed
WackCmds	tips on using Wack

AMICUS Disk 2

C programs:

alib	AmigaDOS object library manager , S-E
ar	text file archive program, S-E
fixobj	auto-chops executable files
shell	simple CLI shell, S-E
sq, usq	file compression programs, S-E
YachtC	a familiar game, S-E
Make	a simple 'make' programming utility, S-E
Emacs	an early version of the Amiga text editor, S-E-D

Assembler programs:

bsearch.asm	binary search code
qsort.asm	Unix compatible qsort() function, source and C test program
setjmp.asm	setjmp() code for Lattice 3.02
SVprintf	Unix system V compatible printf()
trees.o	Unix compatible tree() function, O-D

(This disk formerly had IFF specification files and examples. Since this spec is constantly updated, the IFF spec files have been moved to their own disk in the AMICUS collection. They are not here.)

John Draper Amiga Tutorials:

Animate	describes animation algorithms
Gadgets	tutorial on gadgets
Menus	learn about Intuition menus

AMICUS Disk 3

C programs:

Xref	a C cross-reference gen., S-E
6bitcolor	extra-half-bright chip gfx demo, S-E
Chop	truncate (chop) files down to size, S-E
Cleanup	removes strange characters from text files
CR2LF	converts carriage returns to line feeds in Amiga files, S-E
Error	adds compile errors to a C file, S
Hello	window ex. from the RKM, S
Kermit	generic Kermit implementation, flakey, no terminal mode, S-E
Scales	sound demo plays scales, S-E
SkewB	Rubik cube demo in hi-res colors, S-E

AmigaBasicProgs(dir)

Automata	cellular automata simulation
CrazyEights	card game
Graph	function graphing programs
WitchingHour	a game

ABasiC programs:

Casino	games of poker, blackjack, dice, and craps
Gomoku	also known as 'othello'
Sabotage	sort of an adventure game

Executable programs:

Disassem	a 68000 disassembler, E-D
DpSlide	shows a given set of IFF pictures, E-D
Arrange	a text formatting program, E-D

Assembler programs:

Argoterm	a terminal program with speech and Xmodem, S-E

AMICUS Disk 4 Files from the original Amiga Technical BBS

Note that some of these files are old, and refer to older versions of the operating system. These files came from the Sun system that served as Amiga technical support HQ for most of 1985. These files do not carry a warranty, and are for educational purposes only. Of course, that's not to say they don't work.

Complete and nearly up-to-date C source to 'image.ed', an early version of the Icon Editor. This is a little flaky, but compiles and runs.

An Intuition demo, in full C source, including files: demomenu.c, demomenu2.c, demoreq.c, getascii.c, idemo.c, idemo.guide, idemo.make, idemoall.h, nodos.c, and txwrite.c

addmem.c	add external memory to the system
bobtest.c	example of BOB use
consoleIO.c	console IO example
creaport.c	create and delete ports
creastdi.c	create standard I/O requests
creatask.c	creating task examples
diskio.c	example of track read and write
dotty.c	source to the 'dotty window' demo
dualplay.c	dual playfield example
flood.c	flood fill example
freemap.c	old version of 'freemap'
geltools.c	tools for VSprites and BOBs
gfxmem.c	graphic memory usage indicator
hello.c	window example from RKM
inputdev.c	adding an input handler to the input stream
joystik.c	reading the joystick
keybd.c	direct keyboard reading
layertes.c	layers examples
mousport.c	test mouse port
ownlib.c, ownlib.asm	example of making your own library with Lattice
paratest.c	tests parallel port commands
seritest.c	tests serial port commands
serisamp.c	example of serial port use
prinintr.c	sample printer interface code
prtbase.h	printer device definitions
regintes.c	region test program
setlace.c	source to interlace on/off program
setparallel.c	set the attributes of the parallel port
SetSerial.c	set the attributes (parity, data bits) of the serial port
singplay.c	single playfield example
speechtoy.c	source to narrator and phonetics demo
timedely.c	simple timer demo
timer.c	exec support timer functions
timrstuf.c	more exec support timer functions
WhichFont.c	loads and displays all available system fonts

process.i and prtbase.i assmebler include files:

autorqstr.txt	warnings of deadlocks with autorequesters
consoleIO.txt	copy of the RKM console I/O chapter
diskfont.txt	warning of disk font loading bug
fullfunc.txt	list of #defines, macros, functions
inputdev.txt	preliminary copy of the input device chapter

License information on Workbench distribution license printer pre-release copy of the chapter on printer drivers, from RKM 1.1 v11fd.txt 'diff' of .fd file changes from version 1.0 to 1.1 v28v1.diff 'diff' of include file changes from version 28 to 1.0

AMICUS Disk 5 Files from the Amiga Link / Amiga Information Network

Note that some of these files are old, and refer to older versions of the operating system. These files are from Amiga Link. For a time, Commodore supported Amiga Link, aka AIN, for online developer technical support. It was only up and running for several weeks. These files do not carry a warranty, and are for educational purposes only. Of course, that's not to say they don't work.

A demo of Intuition menus called 'menudemo', in C source

whereis.c	find a file searching all subdirectories
bobtest.c	BOB programming example
sweep.c	sound synthesis example

Assembler files:

mydev.asm	sample device driver
mylib.asm	sample library example
mylib.i	
mydev.i	
asmsupp.i	
macros.i	assembler include files

Texts:

amigatricks	tips on CLI commands
extdisk	external disk specification
gameport	game port spec
parallel	parallel port spec
serial	serial port spec
v1.1update	list of new features in version 1.1
v1.1h.txt	'diff' of include file changes from version 1.0 to 1.1

Files for building your own printer drivers, including dospecial.c, epsondata.c, init.asm, printer.c, printer.link, printertag.asm, render.c, and wait.asm. This disk does contain a number of files describing the IFF specification. These are not the latest and greatest files, but remain here for historical purposes. They include text files and C source examples. The latest IFF spec is elsewhere in this library.

AMICUS Disk 6 IFF Pictures

This disk includes the DPSlide program, which can view a given series of IFF pictures, and the 'showpic' program, which can view each file at the click of an icon, and the 'saveilbm' program, to turn any screen into an IFF picture. The pictures include a screen from ArticFox, a Degas dancer, the guys at Electronic Arts, a gorilla, horses, King Tut, a lighthouse, a screen from Marble Madness, the Bugs Bunny Martian, a still from an old movie, the Dire Straits moving company, a screen from Pinball Contruction Set, a TV newcaster, the PaintCan, a world map, a Porsche, a shuttle mission patch, a tyrannosaurus rex, a planet view, a VISA card, and a ten-speed.

AMICUS Disk 7 DigiView HAM demo picture disk

This disk has pictures from the DigiView hold-and-modify video digitizer. It includes the ladies with pencils and lollypops, the young girl, the bulldozer, the horse and buggy, the Byte cover, the dictionary page, the robot and Robert. This includes a program to view each picture separately, and all together as separate, slidable screens.

AMICUS Disk 8

C programs:

Browse	view text files on a disk, using menus S-E-D
Crunch	removes comments and white space from C files, S-E
IconExec	EXECUTE a series of commands from Workbench S-E
PDScreen Dump	dumps Rastport of highest screen to printer
SetAlternate	sets a second image for an icon, when clicked once S-E
SetWindow	makes windows for a CLI program to run under Workbench S-E
SmallClock	a small digital clock that sits in a window menu bar
Scrimper	the screen printer in the fourth Amazing Computing, S-E

Amiga Basic Programs:

(Note: Many of these programs are present on AMICUS Disk 1. Several of these were converted to Amiga Basic, and are included here.)

AddressBook	a simple address book database
Ball	draws a ball
Cload	program to convert Compuserve hex files to binary, S-D
Clue	the game, Intuition driven
ColorArt	art drawing program
DeluxeDraw	the drawing program in the 3rd issue of Amazing Computing, S-D
Eliza	conversational computer psychologist
Othello	the game, as known as 'go'
RatMaze	3D ratmaze game
ROR	boggling graphics demo
Shuttle	draws 3D pictures of the space shuttle
Spelling	simple spelling program
YoYo	wierd zero-gravity yo-yo demo, tracks yo-yo to the mouse

Executable programs:

3Dcube	Modula-2 demo of a rotating cube
AltIcon	sets a second icon image, displayed when the icon is clicked
AmigaSpell	a slow but simple spelling checker, E-D
arc	the ARC file compression program, must-have for telecom, E-D
Bertrand	graphics demo
disksalvage	a program to rescue trashed disks, E-D
KwikCopy	a quick but nasty disk copy program: ignores errors, E-D
LibDir	lists hunks in an object file E-D
SaveILBM	saves any screen as an IFF picture E-D ??
ScreenDump	shareware screen dump program, E only
StarTerm	version 2.0, term program, Xmodem E-D

Texts:

LatticeMain	tips on fixing _main.c in Lattice
GDiskDrive	make your own 5 1/4 drive
GuruMed	explains the Guru numbers
Lat3.03bugs	bug list of Lattice C version 3.03
MForgeRev	user's view of the MicroForge hard drive
PrintSpooler	EXECUTE-based print spooling program

.BMAP files:

These are the necessary links between Amiga Basic and the system libraries. To take advantage of the Amiga's capabilities in Basic, you need these files. BMAPs are included for 'clist', 'console', 'diskfont', 'exec', 'icon', 'intuition', 'layers', 'mathffp', mathieeedoubas', 'mathieeesingbas', 'mathtrans', 'potgo', 'timer' and 'translator'.

AMICUS Disk 9

Amiga Basic Programs:

FlightSim	simple flight simulator program
HuePalette	explains Hue, Saturation, and Intensity
Requester	ex. of doing requesters from Amiga Basic
ScrollDemo	demonstrates scrolling capabilities
Synthesizer	sound program
WorldMap	draws a map of the world

Executable programs:

BoingI	latest BoingI demo,with selectable speed, E
Brush2C	converts an IFF brush to C data instructions, initialization code, E
Brush2Icon	converts IFF brush to an icon, E
Dazzle	graphics demo, tracks to mouse, E
DeciGEL	assembler program for stopping 68010 errors, S-E-D
Klock	menu-bar clock and date display, E
life	the game of life, E
TimeSet	Intuition-based way to set the time and date,
EMEmacs	another Emacs, more oriented to word processing, S-E-D
MyCLI	a CLI shell, works without the Workbench, S-E-D

Texts:

FnctnKeys	explains how to read function keys from Amiga Basic
HackerSln	explains how to win the game 'hacker'
Ist68010	guide to installing a 68010 in your Amiga
PrinterTip	tips on sending escape sequences to your printer
StartupTip	tips on setting up your startup-sequence file
XfrmrReview	list of programs that work with the Transformer

Printer Drivers:

Printer drivers for the Canon PJ-1080A, the C Itoh Prowriter, an improved Epson driver that eliminates streaking, the Epson LQ-800, the Gemini Star-10, the NEC 8025A, the Okidata ML-92, the Panasonic KX-P10xx family, and the Smith-Corona D300, with a document describing the installation process.

AMICUS Disk 10 Instrument sound demos

This is an icon-driven demo, circulated to many dealers. It includes the sounds of an acoustic guitar, an alarm, a banjo, a bass guitar, a boink, a calliope, a car horn, claves, water drip, electric guitar, a flute, a harp arpegio, a kickdrum, a marimba, a organ minor chord, people talking, pigs, a pipe organ, a Rhodes piano, a saxophone, a sitar, a snare drum, a steel drum, bells, a vibrophone, a violin, a wailing guitar, a horse whinny, and a whistle.

AMICUS Disk 11

C programs

dirutil	Intuition-based, CLI replacement file manager, S-E
cpri	shows and adjusts priority of CLI processes, S-E
ps	shows info about CLI processes, S-E
vidtex	displays Compuserve RLE pictures, S-E

AmigaBasic programs

pointered	pointer and sprite editor program
optimize	optimization ex ample from AC article
calendar	large, animated calendar, diary and date book program
amortize	loan amortizations
brushtoBOB	converts small IFF brushes to AmigaBasic BOB OBJECTS
grids	draw and play waveforms
hilbert	draws Hilbert curves
madlib	mad lib story generator
mailtalk	talking mailing list program
meadows3D	3D graphics program, from Amazing Computing™ article
mousetrack	mouse tracking example in hires mode
slot	slot machine game
tictactoe	the game
switch	pachinko-like game
weird	makes strange sounds

Executable programs

cp	unix-like copy command, E
cls	screen clear, S-E
diff	unix-like stream editor uses 'diff' output to fix files
pm	chart recorder performances indicator

Assembler programs

cls	screen clear and CLI arguments example

Modula-2

trails	moving-worm graphics demo
caseconvert	converts Modula-2 keywords to uppercase

Forth

Breshehan	circle algorithm example

Analyze

	12 templates for the spreadsheet Analyze!

There are four programs here that read Commodore 64 picture files. They can translate Koala Pad, Doodle, Print Shop and News Room graphics to IFF format. Of course, getting the files from your C-64 to your Amiga is the hard part.

AMICUS Disk 12

Executable programs

blink	'alink' compatible linker, but faster, E-D
clean	spins the disk for use with disk cleaners, E-D
epsonset	sends Epson settings to PAR: from menu, E-D
showbig	view hi-res pictures in low-res superbitmap, E-D
speaktime	tell the time, E-D
undelete	undeletes a file, E-D
cnvapldhm	converts Apple][low, medium and high res pictures to IFF, E-D
menued	menu editor produces C code for menus, E-D
quick	quick disk-to-disk nibble copier, E-D
quickEA	copies Electronic Arts disks, removes protection, E-D
txed 1.3	demo of text editor from Microsmiths, E-D

C programs

spin3	rotating blocks graphics demo, S-E-D
popcli	start a new CLI at the press of a button, like Sidekick, S-E-D
vsprite	VSprite example code from Commodore, S-E-D
AmigaBBS	Amiga Basic bulletin board program, S-D

Assembler programs

star10	makes star fields like Star Trek intro,S-E-D
Pictures	
Mount Mandelbrot	
	3D view of Mandelbrot set
Star Destroyer	hi-res Star Wars starship
Robot	robot arm grabbing a cylinder
Texts	
vendors	list of Amiga vendors, names, addresses
cardco	fixes to early Cardco memory boards
cinclude	cross-reference to C include files, who includes what
mindwalker	clues to playing the game well
slideshow	make your own slideshows from the Kaleidoscope disk

AMICUS Disk 13

Amiga Basic programs

Routines from Carolyn Scheppner of CBM Tech Support, to read and display IFF pictures from Amiga Basic. With documentation. Also included is a program to do screen prints in Amiga Basic, and the newest BMAP files, with a corrected ConvertFD program. With example pictures, and the SaveILBM screen capture program.

Routines to load and play FutureSound and IFF sound files from Amiga Basic, by John Foust for Applied Visions. With documentation and C and assembler source for writing your own libraries, and interfacing C to assembler in libraries. With example sound.

Executable programs

gravity	Sci Amer Jan 86 gravitation graphic simulation, S-E-D
Texts	
MIDI	make your own MIDI instrument interface, with documentation and a hi-res schematic picture.

AMICUS Disk 14

Several programs from Amazing Computing issues:

Tools	
Dan Kary's	C structure index program, S-E-D
Amiga Basic programs	
BMAP Reader	by Tim Jones
IFFBrush2BOB	by Mike Swinger
AutoRequester	example
DOSHelper	Windowed help system for CLI commands, S-E-D
PETrans	translates PET ASCII files to ASCII files, S-E-D
C Squared	Graphics program from Scientific American, Sept 86, S-E-D
crlf	adds or removes carriage returns from files, S-E-D
dpdecode	decrypts Deluxe Paint, removes copy protection, E-D
queryWB	asks Yes or No from the user, returns exit code, S-E
vc	VisiCalc type spreadsheet, no mouse control, E-D
view	views text files with window and slider gadget, E-D

Oing, Sproing, yaBoing, Zoing are sprite-based Boing! style demos, S-E-D

CLIClock, sClock, wClock are window border clocks, S-E-D

Texts

An article on long-persistance phospor monitors, tips on making brushes of odd shapes in Deluxe Paint, and recommendations on icon interfaces from Commodore-Amiga.

AMICUS 15

The C programs include:

'pr'	a file printing utility, which can print files in the background, and with line numbers and control character filtering.
'fm'	displays a chart of the blocks allocated on a disk.
'Ask'	questions an 'execute' file, returns an error code to control the execution in that batch file
'Stat'	an enhanced version of AmigaDOS 'status' command.
'Dissolve'	random-dot dissolve demo displays IFF picture slowly, dot by dot, in a random fashion.
'PopCLI2'	invoke new CLI window at the press of a key.

The executable programs include:

'Form'	file formatting program through the printer driver to select print styles
'DiskCat'	catalogs disks, maintains, sorts,merges lists of disk files
'PSound'	SunRize Industries' sampled sound editor & recorder
'Iconmaker'	makes icons for most programs
'Fractals'	draws great fractal seascapes and mountainscapes.
'3D Breakout'	3D glasses, create breakout in a new dimension
'AmigaMonitor'	displays lists of open files, memory use, tasks, devices and ports in use.
'Cosmoroids'	version of the 'asteroids' for the Amiga.
'Sizzlers'	high resolution graphics demo written in Modula 2.

Texts:

'ansi.txt'	explains escape sequences the CON: device responds to.
'FKey'	includes template for making paper to sit in the tray at the top of the Amiga keyboard.
'Spawn'	programmer's document from Commodore Amiga, describs ways to use the Amiga's multitasking capabilities in your own programs.

AmigaBasic programs:

'Grids'	draw sound waveforms, and hear them played.
'Light'	a version of the Tron light-cycle video game.
'MigaSol'	a game of solitaire.
'Stats'	program to calculate batting averages
'Money'	"try to grab all the bags of money that you can."

AMICUS 15 also includes two beautiful IFF pictures, of the enemy walkers from the ice planet in Star Wars, and a picture of a cheetah.

AMICUS 16

'juggler'	demo by Eric Graham. a robot juggler bouncing three mirrored balls, with sound effects. Twenty-four frames of HAM animation are flipped quickly to produce this image. You control the speed of the juggling. The author's documentation hints that this program might someday be available as a product.

IFF pictures

parodies of the covers of Amiga World and Amazing Computing magazines.

C programs:

'Inputhandler'	example of making an input handler.
'FileZap3'	binary file editing program
'ShowPrint'	displays IFF picture, and prints it.
'Gen'	program indexes and retrieves C structures and variables declared in the Amiga include file system.

Executable Programs:

'FixHunk2'	repairs an executable program file for expanded memory
'ms2smus'	converts Music Studio files to IFF standard 'SMUS' format. I have heard this program might have a few bugs, especially in regards to very long songs, but it works in most cases.
'Missile'	Amiga version of the 'Missile Command' video game.

This disk also contains several files of scenarios for Amiga Flight Simulator II. By putting one of these seven files on a blank disk, and inserting it in the drive after performing a special command in this game, a number of interesting locations are preset into the Flight Simulator program. For example, one scenario places your plane on Alcatraz, while another puts you in Central Park

AMICUS 17

Telcommunications disk whuch contains six terminal programs.

"Comm" V1.33	term prog. with Xmodem, WXmodem,
"ATerm" V7.2	term prog. includes Super Kermit
"VT-100"V2.6	Dave Wecker's VT-100 emulator with Xmodem,Kermit, and scripting
"Amiga Kermit" V4D(060)	port of the Unix C-Kermit
"VTek" V2.3.1	Tektronix graphics terminal emulator based on the VT-100 prog. V2.3 and contains latest 'arc' file compression
"AmigaHost"V0.9	for Compuserve. Includes RLE graphics abilities & CIS-B file transfer protocol.
"FixHunk"	expansion memory necessity
"FixObj"	removes garbage characters from modem received files
"Txt"	filters text files from other systems to be read by the Amiga E.C.
"addmem"	executeable version for use with mem expansion article in AC v2.3

'arc' file documentation and a basic tutorial on un 'arc'ing files

"arcre"	for makeing "arc" files E.C.

Fred Fish Public Domain Software

Fred Fish Disk 1:

amigademo	Graphical benchmark for comparing amigas.
amigaterm	simple communications program with Xmodem
balls	simulation of the "kinetic thingy" with balls on strings
colorful	Shows off use of hold-and-modify mode.
dhrystone	Dhrystone benchmark program.
dotty	Source to the "dotty window" demo on the Workbench disk.
freedraw	A small "paint" type program with lines, boxes, etc.
gad	John Draper's Gadget tutorial program
gfxmem	Graphical memory usage display program
halfbrite	demonstrates "Extra-Half-Brite" mode, if you have it
hello	simple window demo
lattfp	accessing the Motorola Fast Floating Point l ibrary from C
palette	Sample program for designing color palettes.
trackdisk	Demonstrates use of the trackdisk driver.
requesters	John Draper's requester tutorial and example program.
speech	Sample speech demo program. Stripped down "speechtoy".
speechtoy	Another speech demo program.

Fred Fish Disk 2:

alib	Object module librarian.
cc	Unix-like frontend for Lattice C compiler.
dbug	Macro based C debugging package. Machine independent.
make	Subset of Unix make command.
make2	Another make subset command.
microemacs	Small version of emacs editor, with macros, no extensions
portar	Portable file archiver.
xrf	DECUS C cross reference utility.

Fred Fish Disk 3:

gothic	Gothic font banner printer.
roff	A "roff" type text formatter.
ff	A very fast text formatter
cforth	A highly portable forth implementation. Lots of goodies.
xlisp	Xlisp 1.4, not working correctly.

Fred Fish Disk 4:

banner	Prints horizontal banner
bgrep	A Boyer-Moore grep-like utility
bison	CNU Unix replacement 'yacc', not working.
bm	Another Boyer-Moore grep-like utility
grep	DECUS grep
kermit	simple portable Kermit with no connect mode.
MyCLI	Replacement CLI for the Amiga. Version 1.0
mandel	A Mandelbrot set program, by Robert French and RJ Mical

Fred Fish Disk 5:

cons	Console device demo program with supporting macro routines.
freemap	Creates a visual diagram of free memory
input.dev	sample input handler, traps key or mouse events
joystick	Shows how to set up the gameport device as a joystick.
keyboard	demonstrates direct communications with the keyboard.
layers	Shows use of the layers library
mandelbrot	IFF Mandelbrot program
mouse	hooks up mouse to right joystick port
one.window	console window demo
parallel	Demonstrates access to the parallel port.
printer	opening and using the printer, does a screen dump, not working
print.support	Printer support routines, not working.
proctest	sample process creation code, not working
region	demos split drawing regions
samplefont	sample font with info on creating your own
serial	Demos the serial port
singlePlayfield	Creates 320 x 200 playfield
speechtoy	latest version of cute speech demo
speech.demo	simplified version of speechtoy, with IO requests
text.demo	displays available fonts
timer	demos timer.device use
trackdisk	demos trakodisk driver

Fred Fish Disk 6:

compress	like Unix compress, a file squeezer
dadc	analog clock impersonator
microemacs	upgraded version of microemacs from disk 2
mult	removes multiple occuring lines in files
scales	demos using sound and audio functions
setparallel	Allows changing parallel port parameters
setserial	Allows changing serial port parameters.
sortc	quicksort based sort program, in C
stripc	Strips comments and extra whitespace from C source

Fred Fish Disk 7:

This disk contains the executables of the game Hack, version 1.0.1.

Fred Fish Disk 8:

This disk contains the C source to Hack on disk 7.

Fred Fish Disk 9:

moire	Draws moire patterns in black and white
MVP-FORTH	Mountain View Press Forth, version 1.00.03A. A shareware version of FORTH from Fantasia Systems.
proff	a more powerful text formatting program
setlace	Program to toggle interlace mode on and off.
skewb	a rubic's cube type demo
sparks	moving snake Graphics demo

Fred Fish Disk 10:

conquest	An interstellar adventure simulation game
dehex	convert a hex file to binary
filezap	Patch program for any type of file.
fixobj	Strip garbage off Xmodem transferred files.
iff	Routines to read and write iff format files.
ld	simple directory program
ls	Minimal UNIX ls, with Unix-style wildcarding, in C
sq,usq	file squeeze and unsqueeze
trek73	Star Trek game
yachtc	Dice game.

Fred Fish Disk 11:

dpslide	slide show program for displaying IFF images with miscellaneous pictures

Fred Fish Disk 12:

amiga3d	Shows a rotating 3 dimensional solid "Amiga sign".
ArgoTerm	a terminal emulator program, written in assembler
arrow3d	Shows a rotating 3 dimensional wire frame arrow.
ld4	directory listing program
IconExec	
SetWindow	two programs for launching programs from Workbench that presently only work under CLI.
SetAlternate	Makes an icon show a second image when clicked once
StarTerm	terminal emulator, with ASCII Xmodem, dialer, more.

Fred Fish Disk 13:

A Bundle of Basic programs, including:

Jpad	toybox	ezspeak	mandlebrot
xmodem	3dsolids	addbook	algebra
ror	amgseq1	amiga-copy	band
bounce	box	brickout	canvas
cardfi	circle	colorcircles	Copy
cubes1	cutpaste	date	dogstar
dragon	draw	dynamictriangle	
Eliza	ezterm	fillibuster	fractal
fscape	gomoku	dart	haiku
hal9000	halley	hauntedM	hidden
join	loz	mandel	menu
minipaint	mouse	Orthello	patch
pena	pinwheel	gbox	random-circles
Readme	rgb	rgbtest	Rord
sabotage	salestalk	shades	shapes
shuttle	sketchpad		spaceart
speakspeach	speecheasy		spell
sphere	spiral	striper	superpad
suprshr	talk	terminal	termtest
tom	topography		triangle
wheels	xenos		xmostriper

(note: some programs are Abasic, most are Amigabasic, and some programs are presented in both languages)

Fred Fish Disk 14:

amiga3d	update of #12, includes C source to a full hidden surface removal and 3D graphics
beep	Source for a function that generates a beep sound
dex	extracts text from within C source files
dimensions	demonstrates N dimensional graphics
filezap	update of disk 10, a file patch utility
gfxmem	update of disk 1, graphic memory usage indicator
gi	converts IFF brush files to Image struct, in C text.
pdterm	simple ANSI VT100 terminal emulator, in 80 x 25 screen
shell	simple Unix 'csh' style shell
termcap	mostly Unix compatible 'termcap' implementation.

Fred Fish Disk 15:

Blobs	graphics demo, like Unix 'worms'
Clock	simple digital clock program for the title bar
Dazzle	An eight-fold symmetry dazzler program. Really pretty!
Fish	double buffered sequence cycle animation of a fish
Monopoly	A really nice monopoly game written in AbasiC.
OkidataDump	Okidata ML92 driver and WorkBench screen dump program.
Polydraw	A drawing program written in AbasiC.
Polyfractals	A fractal program written in AbasiC.

Fred Fish Disk 16:

A complete copy of the latest developer IFF disk

Fred Fish Disk 17:

The NewTek Digi-View video digitizer HAM demo disk

Fred Fish Disk 18:

AmigaDisplay	dumb terminal program with bell, selectable fonts
Ash	Prerelease C Shell-like shell program, history, loops, etc.
Browser	wanders a file tree, displays files, all with the mouse
MC68010	docs on upgrading your Amiga to use a 68010
Multidim	rotate an N dimensional cube with a joystick
PigLatin	SAY command that talks in Pig Latin
Scrimper	Screen image printer
Xlisp1.6	source, docs, and executable for a Lisp interpreter.

Fred Fish Disk 19:

BlackJack	text-oriented blackjack game
JayMinerSlides	Slides by Jay Miner, Amiga graphics chip designer, showing flowchart of the Amiga internals, in 640 x 400.
Keymap_Test	test program to test the keymapping routines
LockMon	Find unclosed file locks, for programs that don't clean up.

Fred Fish Disk 20:

AmigaToAtari	converts Amiga object code to Atari format
DiskSalv	program to recover files from a trashed AmigaDOS disk.
Hash	example of the AmigaDOS disk hashing function
Hd	Hex dump utility ala Computer Language magazine, April 86
MandelBrots	Mandelbrot contest winners
MultiTasking	Tutorial and examples for Exec level multitasking
Pack	strips whitespace from C source
PortHandler	sample Port-Handler program that performs. Shows BCPL environment clues.
Random	Random number generator in assembly, for C or assembler.
SetMouse2	sets mouse port to right or left port.
SpeechTerm	terminal emulator with speech capabilities, Xmodem
TxEd	Demo editor from Microsmiths Charlie Heath

Fred Fish Disk 21

This is a copy of Thomas Wilcox's Mandelbrot Set Explorer disk. Very good!

Fred Fish Disk 22

This disk contains two new "strains" of microemacs.

Lemacs	version 3.6 by Daniel Lawrence. For Unix V7, BSD 4.2, Amiga, MS-DOS, VMS. Uses Amiga function keys, status line, execute, startup files, more.
Pemacs	By Andy Poggio. New features include <ALT> keys as Meta keys, mouse support, higher priority, backup files, word wrap, function keys.

Fred Fish Disk 23

Disk of source for MicroEmacs, several versions for most popular operating systems on micros and mainframes. For people who want to port MicroEmacs to their favorite machine.

Fred Fish Disk 24:

Conques	interstaller adventure simulation game
Csh	update to shell on Disk 14, with built in commands,named variables, substitution.
Modula-2	A pre-release version of the single pass Modula-2 compiler originally developed for Macintosh at ETHZ. This code was transmitted to the AMIGA and is executed on the AMIGA using a special loader. Binary only.

Fred Fish Disk 25

Graphic Hack	A graphic version of the game on disks 7 and 8

Fred Fish Disk 25

This is the graphics-oriented Hack game by John Toebes. Only the executable is present.

Fred Fish Disk 26

UnHunk	Processes the Amiga "hunk" loadfiles. Collect code, data, and bss hunks together, allows individual specificatio of code, data, and bss origins, and generates binary file with format reminiscent of Unix "a.out" format. The output file can be easily processed by a separate program to produce Motorola "S-records" suitable for downloading to PROM programmer. By Eric Black.
C-kermit	Port of the Kermit file transfer program and server.
Ps	Display and set process priorities
Archx	Yet another program for bundling up text files and mailing or posting them as a single file unit.

Fred Fish Disk 27

ABdemos	Amiga Basic demos from Carolyn Scheppner.
NewConvertFD	creates .bmaps from fd files.
BitPlanes	finds addresses of and writes to bitplanes of the screen's bitmap.
AboutBmaps	A tutorial on creation and use of bmaps.
LoadILBM	loads and displays IFF ILBM pics.
LoadACBM	loads and displays ACBM pics.
ScreenPrint	creates a demo screen and dumps it to a graphic printer.
Disassem	Simple 68000 disassembler. Reads standard Amiga object files and disassembles the code sections. Data sections are dumped in hex. The actual disassembler routines are set up to be callable from a user program so instructions in memory can be disassembled dynamically. By Bill Rogers.
DvorakKeymap	Example of a keymap structure for the Dvorak keyboard layout. Untested but included because assembly examples are few and far between. By Robert Burns of C-A.
Hypocycloids	Spirograph, from Feb. 84 Byte.
LinesDemo	Example of proportional gadgets to scroll a SuperBitMap.
MemExpansion	Schematics and directions for building your own homebrew 1 Mb memory expansion, by Michael Fellinger.
SafeMalloc	Program to debug 'malloc()' calls
ScienceDemos	Convert Julian to solar and sidereal time, stellar positions and radial velocity epoch calculations and Galilean satellite plotter. By David Eagle.

Fred Fish Disk 28

ABasic games by David Addison: Backgammon , Cribbage, Milestone, and Othello

Cpp	DECUS 'cpp' C preprocessor, and a modified 'cc' that knows about the 'cpp', for Manx C.
Shar	Unix-compatible shell archiver, for packing files for travel.
SuperBitMap	Example of using a ScrollLayer, syncing SuperBitMaps for printing, and creating dummy RastPorts.

Fred Fish Disk 29

AegisDraw Demo	Demo program without save and no docs.
Animator Demo	Player for Aegis Animator files
Cc	Unix-like front-end for Manx C.
Enough	Tests for existance of system resources, files, devices.
Rubik	Animated Rubik's cube program
StringLib	Public domain Unix string library functions.
Vt100	VT-100 terminal emulator with Kermit and Xmodem protocols

Fred Fish Disk 30

Several shareware programs. The authors request a donation if you find their program useful, so they can write more software.

BBS	an Amiga Basic BBS by Ewan Grantham
FineArt	Amiga art
FontEditor	edit fonts, by Tim Robinson
MenuEditor	Create menus, save them as C source, by

	David Pehrson
StarTerm3.0	Very nice telecommunications by Jim Nangano

(Fred Fish Disk#30 is free when ordered with at least three other disks from the collection.)

Fred Fish Disk 31

Life	Life game, uses blitter to do 19.8 generations a second.
Mandelbrot	Version 3.0 of Robert French's program.
MxExample	Mutual exclusion gadget example.
RamSpeed	Measure relative RAM speed, chip and fast.
Set	Replacement for the Manx "set" command for environment variables, with improvements.
Tree	Draws a recursive tree, green leafy type, not files.
TxEd	Crippled demo version of Microsmith's text editor, TxEd.
VDraw	Full-featured drawing program by Stephen Vermeulen.
Xicon	Invokes CLI scripts from icon
Ticon	Displays text files from an icon.

Fred Fish Disk 32

Address	Extended address book written in AmigaBasic.
Calendar	Calendar/diary program written in AmigaBasic.
DosPlus1	First volume of CLI oriented tools for developers.
DosPlus2	Second volume of CLI oriented tools for developers.

Executables only:

MacView	Views MacPaint pictures in Amiga low or high res, no sample pictures, by Scott Evernden.
Puzzle	Simulation of puzzle with moving square tiles.
ShowHAM	View HAM pictures from CLI.
Solitaire	ABasiC games of Canfield and Klondike, from David Addison.
Spin3	Graphics demo of spinning cubes, double-buffered example.
Sword	Sword of Fallen Angel text adventure game written in Amiga Basic.
Trails	Leaves a trail behind mouse, in Modula-2

Fred Fish Disk 33

3dstars	3d version of the "stars" program below.
Bigmap	Low-level graphics example scrolls bitmap with ScrollVPort.
Dbuf.gels	Double-buffered animation example for BOBs and VSprites.
DiskMapper	Displays sector allocation of floppy disks.
MemView	View memory in real time, move with joystick.
Oing	Bouncing balls demo
Sproing	Oing, with sound effects.
ScreenDump	Dumps highest screen or window to the printer.
Sdb	Simple database program from a DECUS tape.
Stars	Star field demo, like Star Trek.
TermPlus	Terminal program with capture, library, function keys, Xmodem, CIS-B protocols.
Vt100	Version 2.0 of Dave Wecker's VT-100 emulator, with scripts and function keys.

Fred Fish Disk 34

Alint	Support files for Gimpel's 'lint' syntax checker
Blink	PD 'alink' compatible linker, faster, better.
Browser	Updated to FF 18 'browser', in Manx, with scroll bars, bug fixes.
Btree	b-tree data structure examples
Btree2	Another version of 'btree'
Calendar	Appointment calendar with alarm.
Less	File viewer, searching, position by percent, line number.
NewFonts	Set of 28 new Amiga fonts from Bill Fischer
Pr	Background print utility, style options, wildcards.
Requester	Deluxe Paint-type file requester, with sample.

Fred Fish Fred Fish Disk 35

ASendPacket	C example of making asynchronous I/O calls to a DOS handler, written by C-A
ConsoleWindow	C example of getting the Intuition pointer a CON: or RAW: window, for 1.2, by C-A.
DirUtil	Walk the directory tree, do CLI operations from menus
DirUtil2	Another variant of Dirutil.
FileRequester	Lattice C file requester module, with demo driver, from Charlie Heath.
MacView	Views MacPaint pictures in Amiga low or high res, with sample pictures, by Scott Evernden.
Plop	Simple IFF reader program
PopCLI	Sidekick-style program invokes a new CLI, with automatic screen blanking.
QuickCopy	Devenport disk copiers duplicate copy-protected disks.
ScrollPf	Dual playfield example, from C-A, shows 400 x 300 x 2 bit plane playfield on a 320 x 200 x 2 plane deep playfield.
SendPacket	General purpose subroutine to send AmigaDos packets.
SpriteMaker	Sprite editor, can save work as C data structure. Shareware by Ray Larson.
Tracker	Converts any disk into files, for electronic transmission. Preserves entire file structure. Shareware by Brad Wilson.
TriClops 3-D	space invasion game, formerly commercial, now public domain. From Geodesic Publications.
Tsize	Print total size of all files in subdirectories.
Unifdef	C preprocessor to remove given #ifdef'd sections of a file, leaving the rest alone. By Dave Yost.
Vttest	VT-100 emulation test program. Requires a Unix system.

Fred Fish Disk 36

Acp	Unix-like 'cp' copy program
Clock	Updated version of clock on disk 15.
Csh	Manx 'csh'-like CLI, history, variables, etc.
DietAid	Diet planning aid organizes recipes, calories
Echo	Improved 'echo' command with color, cursor addressing
FixHunk	Fixs programs to let them run in external memory.
Fm	Maps the sectors a file uses on the disk.
KickBench	Docs, program to make a single disk that works like a Kickstart and Workbench.
Lex	Computes Fog, Flesch, and Kincaid readability of text files.
TunnelVision	David Addison ABasic 3D maze perspective game.
Vc	Visicalc-like spreadsheet calculator program.
Vt100	Version 2.2 of Dave Wecker's telecom program
YaBoing	Oingl style game program shows sprite collision detects

Fred Fish Disk 37

This disk is a port of Timothy Budd's Little Smalltalk system, done by Bill Kinnersley at Washington State University.

Fred Fish Disk 38

CSquared	Sep 86 Sci American, Circle Squared algorithm
FixObj	Strips garbage off Xmodem transfered object files
Handler	AmigaDOS handler (device) example from C-A
Hp-10c	Mimics a HP-10C calculator, written in Modula-2
IFFEncode	Saves the screen as an IFF file
IffDump	Dumps info about an IFF file
Jsh	BDS C-like CLI shell
NewStat	STATUS-like program, shows priority, processes
Reversi	Game of Reversi, version 6.1
UUdecode	Translate binary files to text, Unix-like programs
Vdraw	Drawing program, version 1.14
VoiceFiler	DX MIDI synthesizer voice filer program
Window	Example of creating a DOS window on a custom screen

Fred Fish Disk 39

AnsiEcho	'echo', 'touch', 'list', 'cls' written in assembler.
Display	Displays HAM images from a ray-tracing program, with example pictures.
Driver	Example device driver source, acts like RAM: disk
Xlisp	XLisp 1.7, executable only

Fred Fish Disk 40

Ahost	Terminal emulator with Xmodem, Kermit and CIS B protocols, function keys, scripts, RLE graphics and conference mode.
AmigaMonitor	Dynamically displays the machine state, such as open files, active tasks, resources, device states, interrupts, libraries, ports, etc.
Arc	Popular file compression system, the standard for transitting files
AreaCode	Program that decodes area codes into state and locality.
Blink	'alink' replacement linker, version 6.5
Cosmo	An 'asteriods' clone.
Dg210	Data General D-210 Terminal emulator
DirUtil	Windowed DOS interface program, version 1.4
DOSHelper	Windowed AmigaDOS CLI help program
PagePrint	Prints text files with headers, page breaks, line numbers
PopCLI	Starts a new CLI with a single keystroke, from any program, With a screen-saver feature. Version 2, with source.
SpriteEd	Sprite Editor edits two sprites at a time
X-Spell	Spelling checker allows edits to files

FF 41

AmigaVenture	Create your own text adventure programs in AmigaBasic.
Csh	Version 2.03 of Dillon's Csh-like shell. Executable only
Dbug	Macro based C debugging package, update to FF #2
DualPlayField	example from CBM, update to Intuition manual
GetFile	Heath's file requester, with source
LatXref	Cross reference of Lattice 3.10 header files
Lines	Line drawing demo program
SetFont	Changes font used in a CLI window
Vt100	Version 2.3 of the VT-100 terminal program.

FF 42

This disk contains an Amiga version of MicroGNUEmacs.

FF 43

BasicBoing	AmigaBasic program demos page flipping of a 3D cube
Bbm	Demo copy of B.E.S.T. Business Management System.
BbsList	A list of Amiga Bulletin Board Systems
Cc	C compiler frontends for Manx and Lattice C
Copper	A hardware copper list disassembler
InstIFF	Converts Instruments demo sounds to IFF sampled sounds
PopColours	Adjust RGB colors of any screen
SpriteClock	Simple clock is displayed on a sprite above all screens
ST Emulator	Non-serious Atari ST emulator
WBrun	Lets Workbench programs be run from the CLI
Wild	Two Unix shell style wildcard matching routines

FF 44

Icons	Miscellaneous icons
NewIFF	New IFF material from CBM for sampled voice and music files
RayTracePics	The famous ray-tracing pictures, from FF #39, now converted to IFF HAM format for *much* faster viewing.
ViewILBM	Displays normal and HAM ILBM files

FF 45

Clue	Clue board game
Make	Another 'make', with more features
Pictures	Miscellaneous pictures
Update	Updates an older disk with newer files from another disk
Wherels	Searches a disk for files of given name

FF 46

Asm	Shareware 68010 macro assembler, ROM Kernal Manual compatible
CheckModem	'execute' file program detects presence of modem
Egad	Gadget editor from the Programmers Network
Jive	Transforms a file from English to Jive.
My.lib	A binary only copy of Matt's alternate runtime library. Author: Matt Dillon
ProffMacros	Subset Berkeley 'ms' and 'mm' macros for 'proff'
ValSpeak	Transforms a file from English to Valley Speak.

FF 47

3D-Arm	Simulation of a robotic arm, very good graphics, teaching tool, including C source.
Juggler	Eric Graham's stunning HAM animation of a robot juggler
VT-100	Version 2.4 of Dave Wecker's terminal emulator, with Xmodem and Kermit file transfer protocols

FF 48

Bru	Alpha version of a hard disk file archiver
Comm	Version 1.30 of a terminal emulator with phone directories
Csh	Version 2.04 of Matt Dillion's Unix 'csh'-like CLI replacement, including Lattice and Manx

	C source
Diskperf	Disk benchmark program for Unix and Amiga
Du	Computes disk storage of a file or directory
MemWatch low	Program to watch for programs that trash memory. It attempts to repair the damage, and puts up a requester to inform you of the damage. From the Software Distillery.
Profiler	A realtime execution profiler for Manx C programs. Includes C source.
FF 49	
Cycloids 27	Update of electronic spirograph from disk
DirUtil	Enhanced version of DirUtil from disk 35
MultiDef	Scans a set of object modules and libraries searching for multiply defined symbols
MyUpdate	Disk update utility with options for stripping comments from C header files, and interactive verification of the updating process
Plot	Computes and displays 3 dimensional functions in hires
Polygon	Moire type pattern generator with color cycling
QMouse	Queries whether a mouse button is
pressed.	This can give a return code that can customize a startup-sequence based on whether a mouse button was pressed.
Touch	Example of setting the datestamp on a file, using a new technique from Commodore-Amiga
Trees program	More extensive version of the trees on Disk 31
FF 50	
Asm	Version 1.1 of a shareware 68000 macro assembler, compatible with the Metacomco assembler. This includes an example
startup	module and more Motorola mneumonics.
BreakOut	A brick breakout game, uses 3-D glasses
DiskZap	Version 1.1 of a program to edit disks and binary files
FirstSilicon and	A smart CLI replacement with full editing recall of previous commands
Missile sound,	A Missile Command-type game, with in assembler
PerfectSound	Sound editor for a low-cost sound digitizer
Sizzlers	Graphics demos
UnixArc machines,	Version of 'arc' for Unix System V in C
Wombat	Version 3.01 of Dave Warker's terminal emulator
FF 51	
Bison 4	GNU for Unix 'yacc', working update to disk version
Compress	Update to the file compression program on Disk 6
Cos AmigaBasic	"Wheel of Fortune"-type game in
DifSsed	Unix-like 'dif' and 'ssed' for finding the differences between two files, and then recreating the other, given one file, and the list of differences.
Sq, Usq	Portable versions of the CP/M squeeze and unsqueeze
FF 52	
Assign	Replacement for AmigaDOS 'assign' command in C
Fractal	Makes random fractal terrains
Poly, HAMPoly	Workbench-type demos for making
polygons	in lores and HAM
MxGads	Example of mutual exclusion gadgets with GadgetText
Tek4010	Tektronix 4010 terminal emulator
VDraw	Versions 1.16 and 1.19 of a Deluxe Paint-like drawing program
FF 53	
Animations	Demo animations with player program for Aegis Animator
ARCre	Creates rename scripts for files with long names, so they can be easily 'arc'ed and un'arc'ed.
ARP	Preliminary AmigaDOS replacements for 'break', 'cd', 'chmod', 'echo', 'filenote' and 'makedir'
Compiler	Not fully ported to the Amiga, this is a 68000 C compiler. It wil produce simple assembly language output, but needs a lot of work.
Spreadsheet	Update with source of the 'vc' spreadsheet on disk 36
TarSplit	Port of program to split Unix 'tar' archives
UUencode	Utiltiies to encode and decode binary files for ASCII transmission, expanding them by 35 percent
FF 54	
Hanoi	Solves Towers of Hanoi Problem in it's own Workbench window. by Ali Ozer
ISpell	Port of a Unix screen oriented, interactive spelling checker. (Expansion RAM required) by Pace Willisson
Ing	A Screen of lots of bouncing little windows by Leo 'Bols Ewhac' Schwab
Lav	Displays number of tasks in run queue, averaged over last 1, 5, and 15 minute periods. by Willaim Rucklidge
MIDITools	Programs to play/record through the MIDI I/F. by Fred Cassirer
MoreRows	Program to make the WorkBench Screen larger than normal. by Neil Katin and Jim Mackraz
Tilt	Program to make your Amiga look like it didn't pass vibration testing. by Leo 'Bols Ewhac' Schwab
FF 55	
Csh	V2.05 of Matt Dillon's csh like shell (Modified for Manx C). by Matt Dillon, Modified by Steve Drew
NewStartups	New C Startup modules:
AStartup.asm	with 1.2 fixes and better quote handling.
TWStartup.asm	opens a stdio window, using user specs. by Commodore, posted to BIX by Carolyn Schepper
Palette	Change another program's screen colors. by Carolyn Schepper
PipeDevice	Allows the standard output of one process to be fed to the standard input of another. by Matt Dillon
ScreenSave	Save a normal or HAM mode screen as an IFF file. by Carolyn Schepper
ShanghaiDemo	Demo version of the Activision game Shanghai.
SoundExample	A double buffered sound example for Manx C. by Jim Goodnow
Vsprites	A working vsprite example. by Eric Cotton
Vt100	V2.6 of Dave's Vt100 terminal emulator with kermit and xmodem. by Dave Wecker
FF 56	
ClipBoard	Clipboard device interface routines, to provide a standard interface. by Andy Finkle
ConPackets	Demos the use of DOS Packets, ConUnit, etc. by Carolyn Schepper
GetDisks	Program to find all available disk device names and return them as an exec list. by Philip Lindsay
GetVolume	Program to get volume name of the volume that a given file resides on. by Chuck McManis
Icon2C	Reads an icon file and writes out a fragment of C code with the icon data structures. by Carolyn Schepper
MergeMem	Program to merge the MemList entries of sequentially configured RAM boards. by Carolyn Schepper
mCAD	An object oriented drawing program, V1.1 by Tim Mooney
FF 57	
CutAndPaste	Implementaions of Unix cut and paste commands. by John Weald
GraphIt	Progran to plot simple functions in 2 or 3 dimensions. by Flynn Fishman
Juggler	V1.2 of robot juggler animation. Uses HAM mode and ray tracing. by Eric Graham
MouseReader	Shareware program to read text files and view IFF files using only the mouse. by William Betz
Ogre	Game of tactical ground combat in the year 2086. by Michael Caplinger; Amiga port by Hobie Orris
Splines	Program to demonstrate curve fitting and rendering techniques. by Helene (Lee) Taran
FF 58	
ASDG-rrd	Extremely useful shareware recoverable ram disk. by Perry Kivolowitz
BigView	Displays any IFF picture, independent of the physical display size, using hardware scroll. by John Hodgson
EGraph	Reads pairs of x and y value from a list of files and draws a formatted graph. by Laurenece Turner
HyperBase	Shareware data management system. V1.5
MemClear	Walks through the free memory lists, zeroing free memory along the way. by John Hodgson
NewZAP	A third-generation multi-purpose file sector editing utility. V3.0 by John Hodgeson
RainBow	A Maurauder-Style rainbow generator. by John Hodgson
SMUSPlayers	Two SMUS plays, to play SMUS IFF music formatted files. by John Hodgson
View	A tiny ILBM viewer by John Hodgson
WBdump	JX-80 optimized workbench printer that does not use DumpRPort. by John Hodgson

To Be Continued.....

In Conclusion

To the best of our knowledge, the materials in this library are freely distributable. This means they were either publicly posted and placed in the Public Domain by their Author, or they have restrictions published in their files to which we have adhered. If you become aware of any violation of the author's wishes, please contact us by mail.

•AC•

Please see page 65 for order form.

Index of Advertisers

Support the Amiga™ & Amazing Computing™,

Write!

Your thoughts, experiences, and programs are needed by others. For an Author's guide, write to: Author's Guide, PiM Publications, Inc., P.O.Box 869, Fall River, MA. 02722.

DIGI PAINT

all at one time!

The 4096 Color^ Paint Program for the Amiga™

Create original art with a palette of 4096 colors.

*Load H.A.M. images from **Digi-View™** or 32 color images from DeluxePaint™ and other IFF programs.*

From the creators of **Digi-View** comes **Digi-Paint**, the first paint program to take full advantage of the **Amiga's** exclusive "hold-and-modify" mode. No longer are you limited to 32 colors. With **Digi-Paint**, you can use all 4096 colors on screen simultaneously. Features include brushes, smooth shading, magnify, cut & paste, output to printer, and full IFF load and save. **Digi-Paint** was programmed completely in assembly language for the fastest possible response. Give your **Amiga** the graphics power of systems costing thousands of dollars more. See your **Amiga** dealer today or call toll-free for **Digi-Paint**, the 4096 color paint program.

Only $59.95

Orders Only (800) 358-3079 ext. 342
Customer Service (913) 354-9332

NEWTEK
INCORPORATED
701 Jackson • Suite B3 • Topeka, KS • 66603

An Inventive Approach To Practical Reality

ZING!™

Sometimes you just have to take matters into your own hands. Or hand. More than once, a brainstorm has turned into a shower of practical reality. To the skeptic, the truth can be shocking.

Well, now you can start your AMIGA flying high with the dynamic, new ZING! utility programs from MERIDIAN SOFTWARE, INC.

ZING! programs are the best program companions for the AMIGA on the market today. Separately, they are practical, yet dynamic working partners. Strung together. . .the sky's the limit!

The High Point In Software Innovations. . .

MERIDIAN™ SOFTWARE INC.

P.O. Box 890408 Houston, Texas 77289-0408
(713) 488-2144

ZING! . The fastest and easiest way to work with the AMIGA. ZING! is actually a collection of utility programs which, after installation, are invisible until called up by the user through the use of hot keys. The main goal of ZING!, is to essentially eliminate the need to learn the operating system protocol and cryptic commands. Functions include: copying, editing, deleting, sorting, renaming, searching, reorganizing files, and much more! Shipping now for: $79.95

ZING!Keys ZING!Keys is a sophisticated, reprogrammable MACRO and Hot Key program. A program which can stand on its own, or be tied to ZING! You can train ZING!Keys to accomplish the most annoyingly repetitive tasks in a much easier fashion. Plus, ZING!Keys allows you to retrieve commands and reuse them, as well as the ability to record mouse movements and use them with the press of a key. Shipping now for: $49.95

ZING!Spell As the name implies, ZING!Spell is a program which can detect misspelled words as you are typing them. A simple interface allows you to correct a word, or add a word to the dictionary. You'll never need to buy another spelling checker. Can be used with other ZING! components or alone. $49.95

Please add $3.00 shipping and handling to each order.

Credit cards and dealer inquiries welcome.

MicroSearch Discoveries

"At MicroSearch, we listen to our customers . . . carefully"

City Desk Sets New Standards in Desktop Publishing

MicroSearch is about to set the standard for desktop publishing for the Amiga with its NEW City Desk Desktop publishing program. City Desk was designed, with *you* in mind, to be an integrated package from the start, with the emphasis on exploiting the versatility and simplicity of the Amiga. Written by SunRize Industries of College Station, Texas, (the developers of the digital sound sampler Perfect Sound, *see below)*, the package will be available May 1, 1987. The retail price for City Desk will be $149.95.

Postscript compatability and kerning. Naturally, City Desk will have postscript compatability and kerning. Since these are rapidly becoming the standard in the desktop publishing industry and because you want them, City Desk is written to include these two features from the start.

- **Mix graphics and text on your page**
- **View and edit multiple pages**
- **Any number of columns per page**
- **Mouse moves and crops graphics and text**
- **Mix any number of fonts**
- **Use Laser printer for typeset look**
- **Draw lines and boxes**
- **Clip art included**

Integrate Graphics. With the graphics integration feature, City Desk is designed with the sales and marketing professional in mind as well as the graphics art industry. City Desk is ideal for making catalog updates and announcing price specials without messy cut and paste or the expense of typesetting. City Desk can be used by the graphics art professional for high quality page layout, highlighting easily with lines and boxes you draw yourself.

IFF file transfer for photographs. Integrating graphics is made possible by use of the industry standard IFF file format in the program. This allows you to use any file in IFF brush format, even digitized photographs or the included library of clip art, to get your message across.

We know what your customers want to buy. Use City Desk to make newsletters similar to this one and satisfy your customers. Dealers, your inquiries are invited.

(Photograph has been retouched).

After loading the file, City Desk allows you to place the text or graphics *anywhere* on the page and easily enlarge, shrink, or crop the image or text by using the mouse.

Use other text and multiple fonts. City Desk will also allow you to load text from any of the Amiga word processors that are currently on the market. By using City Desk's powerful imbedded command codes and the Amiga's standard fonts, you can mix any number of different fonts to enhance your document! This lets you have *unprecedented* control of text fonts, sizes and styles.

Multiple pages and columns. With City Desk you can view multiple pages at one time and easily drag text and graphics from one page to another, as well as format any of those pages into as many columns as you feel are necessary. City Desk also lets you justify the text — right, left, or center — while it is in the column.

Laser printer compatability. City Desk lets you use the HP LaserJet+ for crisp typeset quality, or any other Preferences printer.

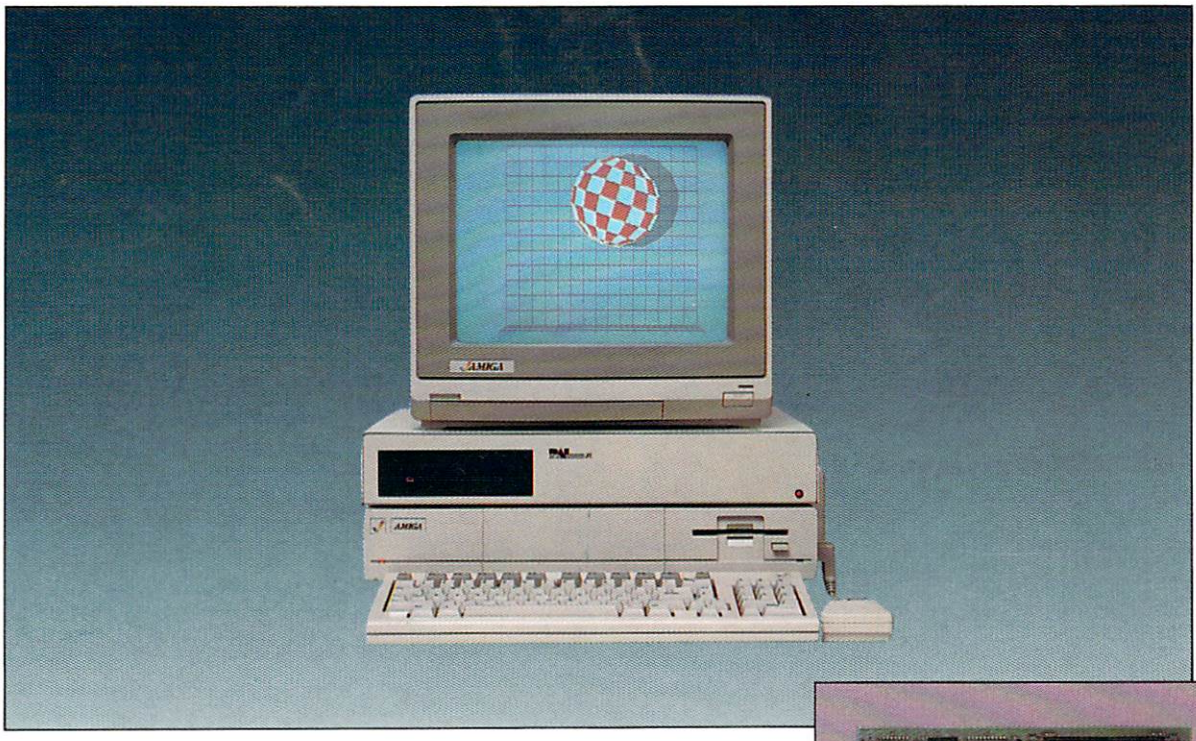
Perfect Sound Records in Stereo

MicroSearch has the Perfect Solution for stereo recording of digitized sounds. Perfect Sound! Other digital sound samplers can record only one channel at a time. Perfect Sound can record in true stereo simultaneously. Because of the popularity of this feature, there are over five thousand Perfect Sounds in use today.

SunRize Industries came up with the idea and they were able to successfully produce the finished product. Now you can flip, graph, insert or delete recorded sounds. Other capabilities allow you to create IFF instruments and change playback or record speeds. Perfect Sound comes with the "C" source code and a library of recorded sounds. Perfect Sound retails for $89.95.

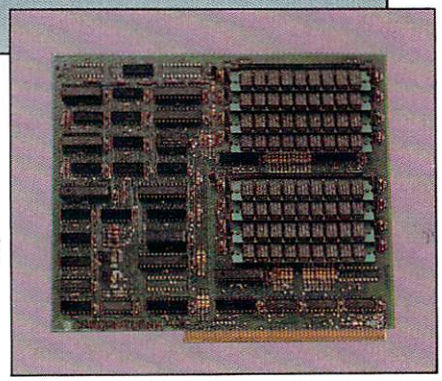
MicroSearch, Inc.
9896 Southwest Freeway
Houston, Texas 77074
(713) 988-2818

There's a slim difference between the ordinary and the extraordinary...PAL JR.



- *1 MByte fast expansion RAM at C00000*
- *No wait states*
- *20 MByte hard disk drive*
- *True DMA controller with SCSI option*
- *Battery backed clock/calendar*
- *Open Zorro expansion slot*
- *Low profile, Amiga-beige case*
- *User expandable to 9 MBytes RAM*
- *Entire system auto-configures*
- *Suggested retail price $1,495.00*

2 MByte RAM CARD



- *High speed memory board*
- *No wait states during access or refresh*
- *Incorporates SIMM technology*
- *User upgradable to 8 MBytes*

BYTE by BYTE™ CORPORATION

Arboretum Plaza II 9442 Capitol of Texas Highway North Suite 150 Austin, TX 78759 (512) 343-4357